CIRRELT

Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation

Collaborative Agent Teams (CAT) for Distributed Multi-Dimensional Optimization

Marc-André Carle
Alain Martel
Nicolas Zufferey

August 2012

CIRRELT-2012-43

Document de travail également publié par la Faculté des sciences de l'administration de l'Université Laval, sous le numéro FSA-2012-009.

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

UNIVERSITÉ LAVAL · UQÀM Université du Québec à Montréal · HEC MONTRÉAL · ÉCOLE POLYTECHNIQUE MONTRÉAL · Université de Montréal

Collaborative Agent Teams (CAT) for Distributed Multidimensional Optimization

Marc-André Carle[1,2,*], Alain Martel[1,2], Nicolas Zufferey[3]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Department of Operations and Decision Systems, 2325, de la Terrasse, Université Laval, Québec, Canada G1V 0A6

[3] HEC, Faculty of Economics and Social Sciences, University of Geneva, Boulevard du Pont-d'Arve 40, 1211 Geneva 4, Switzerland

Abstract. We present a metaheuristic optimization framework based on a Collaborative Agent Teams (CAT) architecture to tackle large-scale mixed-integer optimization problems with complex structures. This framework introduces several conceptual improvements over previous agent teams approaches. We discuss how to configure the three key components of a CAT solver for a particular multidimensional optimization problem: the problem representation, the design of agents and the information sharing mechanisms between agents. The performance of the approach is studied using a multi-period multi-product supply chain network design problem, and implementation issues are discussed.

Keywords. Matheuristic, parallel metaheuristic, A-teams, supply chain network design.

Acknowledgements. This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) grant no DNDPJ 335078-05, by Defence R&D Canada and by Modellium Inc.

Results and views expressed in this publication are the sole responsibility of the authors and do not necessarily reflect those of CIRRELT.

Les résultats et opinions contenus dans cette publication ne reflètent pas nécessairement la position du CIRRELT et n'engagent pas sa responsabilité.

* Corresponding author: Marc-Andre.Carle@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2012

© Copyright Carle, Martel, Zufferey, and CIRRELT, 2012

1. INTRODUCTION

Recent years have seen significant progress in our ability to solve increasingly larger and complex optimization problems. In particular, the capabilities of generic solvers, both commercial and academic, have significantly improved over the last 20 years (Bixby, Rothberg and Gu 2010). Furthermore, problem-specific exact and heuristic solution approaches are improved and refined constantly in order to tackle more challenging optimization problems. Nevertheless, much remains to be done. Some optimization problems are so complex that finding good-quality solutions remains a very challenging task. Moreover, in some contexts such as real-time scheduling or dynamic dial-a-ride problems, the very nature of the problem requires that high-quality solutions are found quickly (Cordeau and Laporte 2007).

Advances in computing technologies such as the widespread availability of multi-core processors (Gorder 2007), the development of rich parallel programming libraries or environments such as MPI or OpenMP, as well as the advances in network architectures (LAN, WAN or grids over the Internet) has decreased the complexity of developing and implementing parallel algorithms. Furthermore, several new algorithmic approaches, often mixing different types of algorithms, have been recently proposed in the literature.

The objective of this paper is to present CAT (Collaborative Agent Teams), an agent-based metaheuristic based on the Asynchronous Teams (A-Teams) paradigm that is designed to tackle complex multi-dimensional optimization problems. We discuss how to design the three key components of a CAT solver for a particular optimization problem: the problem representation along different dimensional views, the design of the agents and the information sharing mechanisms between the agents. Finally, we present an implementation test-case on a complex supply chain network (SCN) design problem; the results obtained demonstrate the method's benefits compared to a generic solver, as well as the effect of specific CAT components on the approach's overall performance.

The rest of the paper is organized as follows. Section 2 presents a literature review of existing research approaches that contributed to the design of CAT. In Section 3, the agent-based metaheuristic is presented. Section 4 details the implementation test case used. Computational results are presented and discussed in Section 5, and Section 6 concludes the paper.

In order to adequately position the literature and present the problem solving approach proposed, a few definitions are provided. In the rest of this paper, the expression "*decision problem*" refers to a real-world issue requiring a solution as perceived by one or several decision-makers. A decision problem can often be expressed qualitatively in terms of a choice between alternative options or opportunities in a given context. An example of a decision problem arising in logistics would be: selecting the depots a company should use, among a set of alternatives, to deliver its products to its customers at minimal cost.

The term "*optimization model*" refers to a mathematical system formulated to represent a view of a decision problem. An optimization model is specified in terms of a set of decision variables and parameters; it incorporates one or more objective functions and a set of constraints. Decision variables can be continuous, binary or integer. Parameter values can be known or random. The mathematical relations in the model can be linear or nonlinear. An optimization model is an abstract reduction of the real-world decision problem that can be solved with an exact or heuristic algorithm. A classical optimization model formulated to capture the essence of the previous decision problem example is the so-called CFLM – *capacitated facility location model* (Balinski 1961).

The terms "*optimization algorithm*" and "*solution method*" are used to refer to programmable procedures developed to generate one or more high-quality solutions for a given optimization model. The Simplex method, branch-and-bound and branch-and-cut algorithms, greedy heuristics and tabu search metaheuristics are all examples of optimization algorithms. Since the CFLM is a mixed-integer program, it can be solved with generic mathematical programming solvers such as CPLEX® or Gurobi®. Several specialized exact and heuristic methods were also elaborated to solve it (Klose and Drexl 2005). Solution methods vary in scope and structure. A *heuristic* can be defined as a basic solution method able to find, in a reasonable amount of time, a "satisfying" solution to the considered optimization model. Optimality is in general not guaranteed. "Metaheuristic [solution methods] can be defined as upper level general methodologies (templates) that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems" (Talbi 2009). In this paper, the term "*metaheuristics*" is used to refer to such methodologies. The reader interested in having guidelines for the efficient design of metaheuristics according to various criteria (e.g., quality, speed, robustness, ease of adaptation, ability to take advantage of the problem structure) is referred to Zufferey (2012).

The term "*solution*" is used to designate a set of values for the decision variables that satisfy all the constraints of a given optimization model. Our aim in this paper is to help solving *complex* decision problems. When considered in their entirety, these problems lead to complex optimization models, i.e. models with a large number of binary, integer or continuous variables representing different types of interrelated decisions, each of these decision types being associated with constraint sets having different sub-structures. The model can be mono- or multi-objective, it may have nonlinear constraints or objective functions, and some of its parameters may be random variables. Two examples of such real-world problems are pulp-and-paper production scheduling (Murthy et al. 1999) and supply chain network design (Carle, Martel and Zufferey 2012).

2. STRATEGIES TO TACKLE COMPLEX PROBLEMS/MODELS

This section describes several algorithmic strategies to solve complex decision problems and the optimization models used to represent them. It also positions their relative strengths in achieving better performance or tackling more complex problems. A general description of the most relevant strategies is provided rather than a technical description of algorithms. In particular, it is shown that several of these strategies are not mutually exclusive; in fact, several methods can be hybridized to tackle the most challenging problems.

2.1. CLASSICAL APPROACHES

Several optimization models are nowadays "easy" to solve, either with the use of a solver such as CPLEX® or Gurobi®. If a solver can provide an optimal solution in a reasonable amount of time, then it makes sense to use this approach rather than to develop specialized optimization algorithms. However, several optimization models are hard to solve using even state-of-the-art solvers, especially when the model is nonlinear or stochastic.

Models with a single category of binary decision variables and few constraint types are often solved to near-optimality in a reasonable amount of time with metaheuristics. The so-called simple (uncapacitated) facility location model is a good example and it can be solved efficiently with a tabu search metaheuristic (Michel and Van Hentenryck 2004). Metaheuristics can usually successfully tackle such problems even if they have a nonlinear objective functions or constraints. Local-search based metaheuristics are successful on these problems because it is straightforward to create a new solution by applying a simple local transformation on a given solution. When multiple types of integer, binary and continuous decision variables are present in the model, these approaches may not be effective.

2.2. PARALLEL ALGORITHMS

Although parallelism has been used in operations research for more than 30 years, recent changes have increased its importance. Most state-of-the-art commercial solvers such as CPLEX® and Gurobi® are now using several processors at a time if authorized to do so. Furthermore, parallel metaheuristics are more and more popular. According to Melab et al. (2006), parallelism [in metaheuristics] allows for improved solution quality and reduction in the resolution time. Among the strategies used, two are especially relevant. The first is related to parallelization of the search: several copies of a given metaheuristic work in parallel, each having its own parameter settings and possibly exchanging solutions synchronously or asynchronously during the search process. The second strategy relates to the parallelization of some of the most computationally-intensive tasks of the search process (typically solution quality evaluation or neighborhood exploration).

2.3. HYBRIDIZATION

A popular approach when dealing with complex optimization models is to use hybrid methods. According to Talbi (2002), hybridization refers to the combination of different types of algorithms into a single methodology. Of all possible hybridizations, two are especially relevant to this paper. The first results from the combination of two or more metaheuristics, in the hope that one method's strengths compensate for the other method's weaknesses (and vice versa). Raidl, Puchinger And Blum (2010) observe that "well-designed metaheuristic hybrids often perform substantially better than their "pure" counterparts". Several hybridization strategies can be designed for any two given metaheuristics, resulting in a larger number of potential solution methods. A recent review on hybrid metaheuristics is found in Blum et al. (2011). An example of this type of hybridization is memetic algorithms (Moscato and Cotta 2010).

The second type of hybridization is the combination of metaheuristics with exact methods such as branch-and-bound; these hybrid solution methods are often labelled as *matheuristics* (Maniezzo, Stützle and Voss 2009). This class of algorithms effectively combine the ability of metaheuristics to handle a large number of binary or integer decision variables, with the LP- or MIP-based methods' ability to handle a large number of constraints and continuous decision variables. A recent review and classification of this literature is available in Raidl and Puchinger (2008).

2.4. DECOMPOSITION AND MODEL-BASED STRATEGIES

Another family of methods use the optimization model's formulation in order to break it down into smaller and hopefully easier problems. Since the 1960s, decomposition-based solution methods such as Dantzig-Wolfe (dual) (Dantzig and Wolfe 1960) and Benders' (primal) (Benders 1962) decomposition are particularly effective at solving large optimization models that exhibit a specific model structure (such as a block-diagonal parameter matrix structure). The efficiency of these methods often lies in clever reformulation of the optimization model and the availability of a sub-model that can be solved very fast. However, when the decomposed sub-models they yield are themselves very difficult to solve, these methods may not perform well.

Multilevel techniques are another family of methods making use of the model formulation. According to Blum et al. (2011), multilevel techniques start from the optimization model, then iteratively and recursively generates a smaller and smaller model by coarsening until a relatively small model is obtained, creating a hierarchy of optimization models. A solution to the smallest model is found by some optimization algorithm. Then, the solution to this problem is successively transformed into a solution to the model of the next level until a solution to the original optimization model is found. This approach has been successfully applied to traveling salesman, graph coloring (Walshaw 2004), graph partitioning (Toulouse, Thulasiraman and Glover 1999), vehicle routing (Rodney, Soper and Walshaw 2007) and multi-commodity network design (Crainic, Li and Toulouse 2006) models.

In recent years, a number of progressive variable fixing solution methods have been proposed to solve complex models. Several examples of these optimization algorithms exist. The simplest, the LP-rounding strategy (Melo, Nickel and Saldanha-da-Gama 2012), uses a solver to obtain the LP relaxation of the model. The values of integer and binary decision variables that are fractional in the LP relaxation are then rounded in order to obtain an integer-feasible solution. Another example is the progressive variable fixing strategy: a sequence of linear relaxations of the original optimization model is solved, and as many binary and integer variables as possible are fixed at every iteration (Thanh Péton and Boste 2010). These methods are effective to solve problems with a small number of binary and a large number of continuous decision variables.

Model-based strategies are also present in metaheuristics. Variable Space Search (Hertz, Plumettaz and Zufferey 2008) uses three different optimization models to represent the decision problem and a specific algorithm to solve each of them, but the algorithms are used sequentially.

2.5. DISTRIBUTED DECISION MAKING AND AGENT-BASED OPTIMIZATION

Another recent strategy to cope with model complexity is to work at the decision problem level rather than directly on the optimization model. Often, the decision problem can be partitioned into two or more interconnected sub-problems under the responsibility of distinct organisational units. For each of these sub-problems, a sub-model is formulated and solved using a specific optimization method. This approach has several advantages. According to Schneeweiss (2003), "distributed decision making can be useful in order to better understand or manipulate a complex decision situation." This approach is even more suited to decision problems in which multiple decision-makers are involved such as in contract design.

Multi-agent systems (MAS) and agent-based optimization algorithms have also been used recently to model and analyse complex decision problems. Typically, MAS formalize complex decision problems as networks of simpler decision problems; each of these problems being tackled by a separate agent (Schneeweiss 2003). Depending on the degree of sophistication of the approach, the agent may use basic decision rules to make decisions, or formulate an optimization model which is then solved with an appropriate (exact or heuristic) optimization algorithm. A relevant example of this approach is Asynchronous agent teams (A-Teams) (Talukdar, Murthy and Akkiraju 2003), a cooperative MAS used to solve pulp and paper production planning (Murthy et al. 1999, Keskinocak et al. 2002), job-shop scheduling (Aydin and Fogarty 2004), probe selection (Meneses, Pardalos and Ragle 2008), and resource-constrained project scheduling (Ratajczak-Ropel 2010) problems. Other approaches, such as MacDS (Gaudreault, Frayret and Pesant 2009), can be used to model competitive contexts.

2.6. Towards an integrated optimization framework

Several approaches have integrated several strategies outlined above to solve increasingly complex optimization models. Their respective strengths are often complementary: a multi-agent system is indeed well suited to implement parallel and potentially hybrid optimization algorithms. A hybrid matheuristic could and should couple two algorithms working in parallel rather than sequentially. Despite these advantages, very few tools have been proposed to combine the strengths from all these strategies into a single solution system. The following elements should be present in an optimisation framework designed to tackle extremely complex decision problems:

- Drawing inspiration from both the decision problem and alternative optimization model formulations to design adapted solution methods, instead of only one perspective;
- An ingenious use of partitioning strategies, through organisational decomposition (at the problem level) or mathematical decomposition (at the model level), while working on each partition simultaneously in parallel;
- Using the type of optimization algorithm that works best for each sub-model (hybridization and specialization);
- An effective way of sharing information and solutions between different optimization strategies;
- Combination of high-quality solutions from sub-models into high-quality solutions to the complete optimization model.

An optimisation framework based on these characteristics is proposed in the following section.

3. CAT as an agent-based metaheuristic

In this section, we propose CAT (for Collaborative Agent Team) a hybrid distributed agent-based metaheuristic to solve complex decision problems, and associated optimization models, that cannot be efficiently addressed using classical metaheuristics or mathematical decomposition methods. The approach builds on the Asynchronous Teams (A-Teams) paradigm (Talukdar, Murthy and Akkiraju

2003), and it relies on the foundations outlined in section 3.6. The location-routing problem (LRP) (Min, Jayaraman and Srivastava 1998, Nagy and Salhi 2007) is used to illustrate CAT concepts. The LRP involves decisions on the number and location of distribution centers (DCs) in order to serve customers at minimum cost, as well as finding the optimal delivery schedules and vehicle routes to serve these customers.

According to Talukdar, Murthy and Akkiraju (2003), “an asynchronous team is a team of software agents that cooperate to solve a problem by dynamically evolving a shared population of solutions”. Software agents are autonomous; they incorporate their own representation of the problem to be solved, as well as rules to choose when to work, what to work on and when to stop working. As noted by Murthy et al. (1999), the approach is naturally suited to implement multiple representations of a problem, such as advocated in the previous sections. Previous work suggests that A-Teams can host a large variety of optimization algorithms: while Murthy et al. (1999) used simple heuristics as well as linear and integer programming, recent applications such as Aydin and Fogarty (2004) and Ratajczak-Ropel (2010) employ metaheuristics such as tabu search and path relinking. When facing complex optimization models, it makes sense to use the best tools available to tackle each model or sub-model. A multi-agent system allows for that much flexibility.

3.1. Problem solving approach

The following steps are required to solve optimization problems with CAT:

1. Identifying different relevant points of view (dimensions) to examine the decision problem;
2. Formulating optimization models and sub-models for these dimensional views;
3. Designing optimization algorithms to solve each sub-model;
4. Designing optimization algorithms to integrate solutions from sub-models into solutions of the complete optimization models.

These steps are explained in the following sub-sections. That being said, the design of a CAT metaheuristic for a specific decision problem is a complex and iterative task that cannot be completely reduced to a simple step-by-step procedure that guarantees results. As pointed out by Aydin and Fogarty (2004), it is not trivial to design a team of optimization agents that can cooperate effectively. Figure 1 displays the different problem solving constructs used in CAT. Each of them is described in the following subsections. Advice on the number of sub-models and optimization algorithms to elaborate is provided in section 4.2.

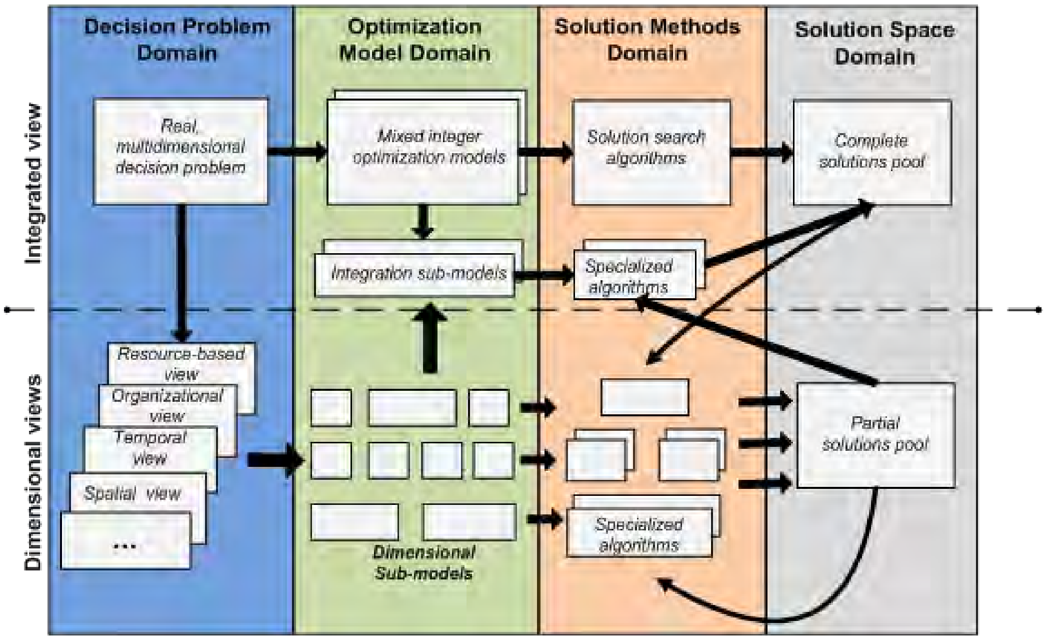

Figure 1: CAT Problem Solving Constructs

3.1.1. Views and sub-problems

Most complex decision problems can be analyzed from different points of view, referred to simply as "views" in this paper. In an intuitive sense, a view is a filter or a lens which emphasizes, reduces or reshapes some aspects of the decision problem to be solved. It often reflects the perceptions of a stakeholder. The *integrated* view refers to a holistic apprehension of the complete decision problem, i.e. one that looks at all relevant facets from a centralized standpoint. Problem solving with CAT requires addressing the problem with an integrated view, as well as with alternative *dimensional views*. Dimensional views are rearrangements of the problem into systems of interrelated *sub-problems*. These sub-problems may cover only a subset of the objectives and decisions of the original problem and they may involve a reduction of some of its facets. Dimensional views are introduced to reduce the complexity of the problem by providing effective partitioning schemes. Dimensional views must be selected before optimization models can be formulated.

A given dimensional view may require the definition of several sub-problems. A sub-problem contains a portion of the decisions and context associated with the decision problem. The number of sub-problems used and the exact definition of each of them are critical to the approach's success. Useful sub-problems possess the following characteristics:

1. They make sense from a business standpoint, i.e. they are easily understandable by a decision-maker. Sub-problems that address some of the decisions related to the job-description of a manager would be a typical example.

2. The set of all the sub-problems associated with a dimensional view must constitute a valid (although sometimes biased) representation of the complete decision problem.

In the LRP context, the following two dimensional views could be defined:

- A *functional* view associated with the types of decisions (location, customer allocation to facilities, and vehicle routing) associated with the decision problem. The problem can then be partitioned into a DC location sub-problem, a customer-to-DC allocation sub-problem, and a transportation or route design sub-problem.
- As with most location problems, the LRP has an inherent *spatial* dimension. Indeed, the customer base served by a company may cover a large territory, and logistics decisions may be made on a national or sales region level instead of globally. In this context, the problem can be partitioned into several regional sub-problems.

These dimensions and the associated sub-problems are easily understandable by a decision-maker. Furthermore, each regional sub-problem contains all decision types, and each functional sub-problem contains decisions for all regions. Consequently, they both constitute a valid representation of the whole problem.

3.1.2. Models and sub-models

The integrated view leads to the formulation of a 'complete' optimization model to represent the decision problem. This model is generally very difficult to solve, but it will be used for a number of purposes. Variants of this model may also be formulated. For each dimensional view, sub-models are formulated to represent sub-problems. These formulations are usually expressed in terms of partitions of complete model decision variable vectors and parameter matrices. They may also be based on alternative modeling formalisms: for example, a constraint programming sub-model could be defined even if the complete optimization model is a mixed-integer program (MIP). A sub-model is useful if it can be solved efficiently. This will usually be the case if the sub-model:

- Corresponds to a generic class of decision models studied in depth in the literature (ex: knapsack, bin packing, graph coloring, traveling salesman, facility location models);
- Can be solved to optimality using generic LP-MIP solvers, or dynamic programming, or simple enumeration (explicit or implicit);
- Isolates a homogeneous group of binary/integer variables and their associated constraints.

3.1.3. Optimization algorithms

Once the sub-models have been formulated, optimization algorithms must be designed to solve them. In CAT, optimization algorithms are implemented as a set of autonomous software agents. Solutions to sub-models are recorded and subsequently used to build complete solutions. The following guidelines may be useful to select a solution method:

- Developing greedy heuristics or GRASP (Feo and Resende 1989) metaheuristics to construct feasible solutions for profit maximizing or cost minimizing sub-models is usually straightforward;
- When the sub-models has been studied in the literature, published solution methods or available code can be integrated into CAT;
- Purely linear sub-models can be solved using a LP-solver library;
- Sub-models involving a homogeneous group of binary/integer variables can usually be solved effectively with a local search metaheuristic since it is rather straightforward to define a neighborhood in this context.

In the LRP context, for example, some of the sub-models formulated and the solution methods selected could be the following:

- A pure facility location sub-model solved wits a MIP solver such as CPLEX;
- A location-allocation sub-model solved with a Lagrangean heuristic (Beasley 1993);
- A vehicle routing sub-model solved with a tabu search heuristic (Cordeau and Laporte 2005);
- A regional LRP sub-model solved with a tabu search / simulated annealing hybrid (Wu, Low and Bai 2002).

3.1.4. Integration sub-models

Solving dimensional sub-models is necessary but not sufficient for a successful CAT implementation. Integration refers to combining the solutions of the sub-models associated with one dimensional view into solutions to the complete optimization model. This is done by solving an integration sub-model heuristically or with exact methods. Integration sub-models are essentially restricted versions of the complete optimization model obtained by fixing the value of several decision variables. The fixed values are provided by the solutions to the dimensional sub-models. By solving the integration sub-model, the optimal value of the non-fixed decision variables is found, and a solution to the complete optimization model is produced. We refer to the set of decision variables to optimize in an integration sub-model as *integration variables*. The integration variables not present in any dimensional sub-models are *linking variables*, and those present in more than one dimensional sub-model are *overlapping variables*.

Integration is also used as a search strategy. For a specific dimensional view, the choices of integration variables lead to different integration sub-models. Several strategies can be used. When the dimensional sub-models solutions are mutually exclusive, as illustrated in Figure 2, then the integration sub-model contains only linking variables, and optimizing these variables provides a feasible solution for the complete model. When the dimensional sub-models solutions are overlapping, a merging integration sub-model such as the one illustrated in Figure 3 is obtained. Since it is rather unlikely that the overlapping variables will have the same value in all partial solutions, the integration sub-model must find the optimal value of these variables. The search space created by a merging sub-model can be further enhanced by including more than one partial solution from a given dimensional sub-model; this adds all the variables from that sub-model to the set of overlapping variables. If the resulting integration sub-model is difficult to solve, one can further constrain the integration sub-model by fixing the values of the overlapping variables that are identical in all partial solutions or restricting the values of the overlapping variables to those found in the partial solutions, resulting in a much smaller model.

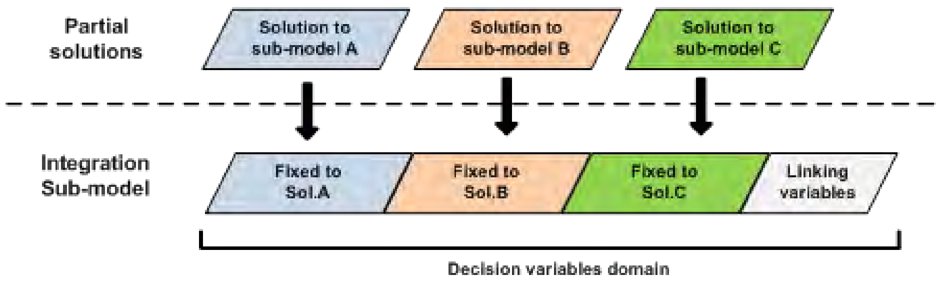

Figure 2: A *Linking* Integration Sub-Model

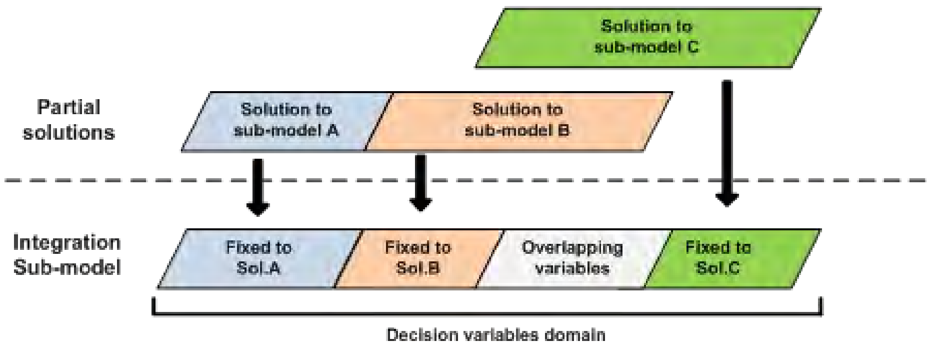

Figure 3: A *Merging* Integration Sub-Model

Depending on the partial solutions chosen for integration, the resulting sub-model may be infeasible. When this occurs, an alternative integration sub-model that seeks to find a feasible solution while keeping most of the partial solutions' characteristics is used. In these sub-model's the original objective function is replaced with the minimization of the number (or amplitude) of decision variable changes when compared with the values found in the sub-problems.

To conclude our LRP example, using the pure location sub-model and the vehicle routing (VRP) sub-model solutions, one would proceed as follows to formulate a merging integration sub-model. The depot location decision variables are fixed using the solution to the pure min-cost location sub-model. Several vehicle routing sub-model solutions are also considered. The resulting integration sub-model selects a set of feasible routes among the routes provided by the VRP sub-models. It is a capacitated set partitioning model for which several solution methods are published in the literature.

3.2. CAT SYSTEM STRUCTURE

In order to solve the decision problem considered, the solution method and solution space constructs illustrated in Figure 1 must be implemented as a multi-agent system. The structure of the CAT system thus obtained is illustrated in Figure 4. It incorporates a blackboard, utility agents and optimization agents. The blackboard acts as a memory and a hub for all communications, and it is the repository of all solutions (to the complete optimization model and to all sub-models). Agents communicate solely through the blackboard interface and do not exchange information directly. New complete or partial solutions are placed on the blackboard and existing solutions are retrieved when necessary. Utility agents provide functionalities required by all agents, such as building mathematical model files for solvers, formatting instance data, as well as compiling solution statistics.

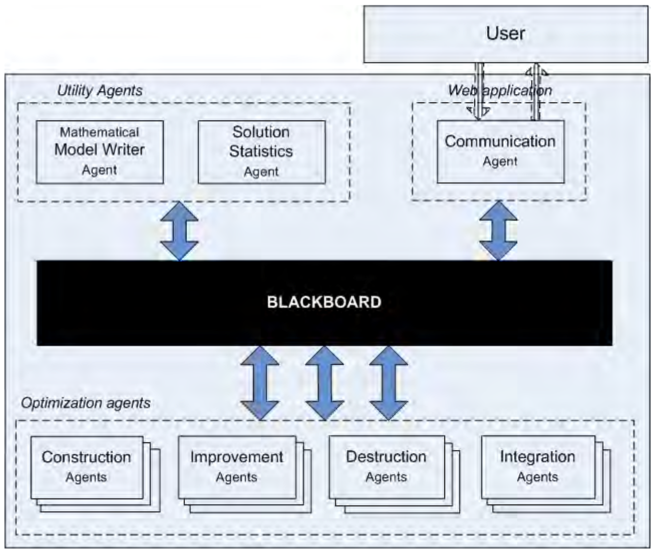

Figure 4: Main Components of a CAT System

The most important agents are of course the optimization agents, which are grouped into four types depending on their role. Construction agents create new solutions from scratch. Improvement agents take existing solutions and modify them to improve their quality. Destruction agents remove unwanted solutions from the repository. Finally, integration agents combine high-quality solutions from two or more dimensional sub-models into solutions to the complete optimization model. These agent roles are defined further in the next section. For the sake of simplicity and clarity, the development of hybrid optimization agents working simultaneously on several sets of skills should be avoided. For instance, if the solution destruction process is not effective enough, it will be difficult to diagnose and correct if it is spread among three agents which also perform solution improvement.

3.3. AGENTS JOB DESCRIPTION

As pointed out by Aydin and Fogarty (2004), a few key questions must be answered when designing a multi-agent optimization system. How many agents should be used? What should their role be? How should they decide when to act, what to act on, and how to act? For all their advantages, agent teams are complex to design and implement. Indeed, if the system uses several algorithms that are similar in nature (simulated annealing variants, for instance) on the same sub-model, it is likely that one of the optimization algorithms (usually the best) will be largely responsible for the team's performance. Also, on a computer with limited resources (memory or processor power), it is likely that adding agents will deteriorate performances. To avoid these pitfalls, Talukdar, Murthy and Akkiraju (2003) advise to start with a small number of agents, and to add new agents with different skills as needed. That being said, according to the literature and to our experience in developing CAT systems, an agent team needs four important basic skills:

1. Quickly obtain feasible solutions to the complete optimization model. Although these may not be of high quality, they provide a basis for other agents to work upon.
2. Improve existing solutions. This can be done at the complete model level or agents can work on specific parts of the problem.
3. Remove unwanted or poor solutions from the population to control its size.
4. Efficiently combine features from solutions originating from different methods or dimensions.

The nature of these skills is discussed in the following sub-sections.

3.3.1. Generating an initial solution using construction agents

Feasible solutions can be obtained easily and quickly with simple heuristics, greedy algorithms, hill-climbing procedures, or even by random generation, for several classes of optimization models. Another option is to use generic LP / MIP heuristics such as feasibility pump variants (Achterberg and Berthold 2007); this approach tends to produce solutions that are very different than those obtained with greedy methods and other heuristics. The key goals at this task are speed and diversity, rather than solution quality. Using a variety of methods usually results in a more diverse initial population of solutions, yielding a higher potential for improvement and collaboration, and reducing the need for specific diversification strategies. If the complete optimization model is difficult to solve but it is easy to find a feasible solution, one can generate solutions to the complete model then infer initial solutions for sub-models from these solutions, thus reducing the number of algorithms and agents needed for this role.

3.3.2. Evolving the solution population using improvement agents

For complex decision problems, it is recommended to work on sub-models rather than on the complete model. Since defining a single neighborhood (or even a set of neighborhoods covering the complete model's range of variables) may be very challenging, local search is typically difficult to use. Evolutionary computing provides generic crossover operators, but solution encoding is complex and on highly constrained problems, developing effective repair functions may be problematic. In order to design a good set of improvement agents, the solution methods used to solve sub-models must be carefully selected. As indicated previously, if the sub-model is a linear program then existing LP-solvers can be used; if it has only one type of binary / integer variable – allowing for the construction of neighborhoods – then a local search metaheuristic (tabu search, VNS, …) can be developed; if it is a variant of a well-studied problem, the best available method can be implemented. It may also be worthy to investigate alternative sub-model reformulations.

A number of strategies can be used to tackle complex sub-models. For instance, an initial solution obtained with a simple heuristic may provide a hot-start for a MIP-solver. Nowadays, commercial solvers incorporate several generic MIP heuristic algorithms such as RINS (Danna, Rothberg and LePape 2005) or local branching (Fischetti and Lodi 2003). When metaheuristics are not efficient, generic MIP heuristics often are. Ball (2011) provides an interesting review of heuristics based on mathematical programming.

To ensure that the system continuously works on each sub-model (or, at least, looks for opportunities to work on it), a dedicated agent should be assigned to its solution. The creation of "super-agents" performing several tasks should be avoided. Such super-agents tend to use too much resource and require complex scheduling rules. As will be shown, simple triggers are easier to manage than complex ones.

3.3.3. Removing poor solutions using a destruction agent

According to Talukdar, Murthy and Akkiraju (2003), solution destruction is as important as solution creation in agent-based optimization. In some situations, the choice of solutions to destroy is obvious, such as when duplicates exist in the population. Aside from maintaining some control over the size of the population, destruction serves two purposes: removing poor quality solutions as well as maintaining diversity in terms of solution characteristics. At the beginning of the search, the solutions in the population are quite diverse. As improvement agents work, the solution quality of the best solutions in the population improves rapidly. At this stage, the destruction agent should focus on removing solutions that are of poor quality. A simple rule such as choosing a solution at random from those in the 4th quartile in terms of solution quality is appropriate.

However, as the overall quality of solution improves, newly created solutions tend not to be competitive in terms of solution quality compared to those which have been improved by several agents. They should have a chance to be improved before they are discarded. Furthermore, as the population improves, working on the same solutions tends to accelerate convergence. As the search progresses, a destruction agent shifts its focus from removing poor solutions to either:

- Removing a random solution which has been improved at least (X – 2) times and is in the bottom half in terms of performance, where X is the number of improvements made on the solution that has been improved most frequently in the population;
- Finding the two solutions in the population that are most similar, and then destroying the worst one;
- Finding the solution which has been used the most frequently to create new solutions among the solutions in the 4th quartile in terms of solution quality, and destroying it.

These rules can be encapsulated in one or more destruction agents, and they work equally well on a population of complete solutions or on a population of partial solutions (solutions to a specific sub-problem). The two metrics necessary to implement them are detailed in section 4.4. Alternatively, some solutions can be “protected” and be immune to deletion for a certain amount of time. These solutions may be the *statu quo* or solutions provided by a decision-maker.

3.3.4. Amalgamation and diversification using integration agents

In our experience, CAT performs better when the population of solutions maintains a high level of diversity. Although the destruction agent works toward maintaining variety, additional diversification strategies may be needed. It is possible to add an agent whose sole objective is to provide the population with radically different solutions than those currently in the population. This agent should maintain a record of what has been proposed in the past, so it does not produce solutions similar to those already removed from the population due to poor solution quality. Two examples of diversification agents are outlined in section 5.2.

The integration of partial solutions from sub-models into complete solutions is a key component of an efficient agent team. At least one optimization algorithm should be provided for each integration sub-model. If two methods are readily available, they can both be used if they generate different high quality solutions. The number of actual agents to use depends on the relative speed at which the improvement agents generate new solutions to sub-models and the amount of computation effort required to solve the integration sub-models.

Integration can be used in very flexible ways. Integration of solutions to sub-models from different dimensional views can be desirable, as long as the resulting merging integration sub-models are not exceedingly difficult to solve. Solving these models often require the design of specific heuristics or the use of a generic heuristic such as local branching (Fischetti and Lodi 2003) or RINS (Danna, Rothberg and LePape 2005). These heuristic are easily implemented using a MIP solver such as CPLEX or Gurobi. This approach is in line with scatter search and path-relinking metaheuristics and is an effective way of reaping the most benefits from using multiple dimensional views. As this type of integration is slightly different than the type of integration sub-models required to assemble complete solutions from partial solutions, these sub-models should be assigned to a different integration agent.

3.4. DECISION RULES AND METRICS

In order to complete its job description, an agent needs formal rules to determine which solution to work on. A trivial option is to select a random solution from the population, but this does not give very good results. Obviously, an agent does not want to select a solution that it has recently worked on. A simple yet effective decision rule is that the agent waits that at least three others agents have improved the solution before attempting to work on it again. Some improvement agents such as local search metaheuristics may want to push that rule a little further: since a local search heuristic explores thoroughly a restricted portion of the search space (Gendreau and Potvin 2010), an agent may want to select a solution that is significantly different than the one it just worked on. In order to design more sophisticated decision rules, a few metrics must be computed and are described below.

3.4.1. Solution ancestry

Agents need an effective way to determine which solutions they recently worked on. In a cooperative context, this information should be accessible to all agents. A simple metric to achieve this objective is solution’s ancestry. Simply put, a solution’s ancestry is its genealogical tree. Each solution keeps track of the solutions used for its creation, or as a basis for its improvement, as well as the agents that worked on it. An improvement agent can then use this information to determine whether it has worked on a solution recently, or on any of its parents. Tied to each solution is a list of agents that have worked on the solution, and whether this attempt at improving the solution succeeded or not. This list is sorted in reverse order. A similar mechanism is used to determine whether or not a solution has transmitted its characteristics to other solutions in the population. Anytime a solution is used to create a new solution or to alter an existing solution, its characteristics are propagated through the population. The new solution is linked to its parent solution(s) through an acyclic directed graph structure, so that it is easy to find all the parents or all offspring of a given solution.

A *propagation index* is also calculated for each solution. This index is set to 0 when the solution is created. When a new solution is created, if this solution has 1 or more parent solutions, it parses its solutions digraph and updates the values of its parents' propagation index in a recursive manner. Let S_0 be the newly created solution, S the solution at the currently active node of the solutions digraph, S_s^p the set of immediate parents of solution S and $\left|S_s^p\right|$ its cardinality, γ_s the current propagation index of solution S, and z the depth of the digraph at the current step of the procedure. The following procedure is used to update the solutions' propagation index:

```
Procedure PropagationIndexUpdate( $S_0$, $z = 0$ )
Get $S_s^p$ of active solutions
Set $z = z + 1$
For each solution $S \in S_s^p$
    If $S_s^p \neq \varnothing$ then
        PropagationIndexUpdate( $S$, $z$ )
        Set $\gamma_s = \gamma_s + \frac{1}{2^{(z-1)}\left|S_s^p\right|}$
    End If
```

For practical reasons, a maximum depth of z = 5 can be fixed in order to avoid a large number of infinitesimal increases in propagation indexes. The larger the value of γ_s, the more solution S has been used in the generation of new solutions.

In the example depicted in Figure 5, the solutions are numbered in the order in which they were generated, so S5 and S4 are the parents of S6, S1 and S2 are the parents of S4, and so on. Since a new solution cannot be the parent of an older one, finding whether two solutions are related requires only parsing the digraph associated with the newest solution. Table 1 presents the results of updating propagation indexes associated with the new solution S6. $\Delta\gamma_s$ lists the increase in propagation index resulting from the creation of solution S6, while Total γ_s lists the propagation index of each solution after the addition of S6.

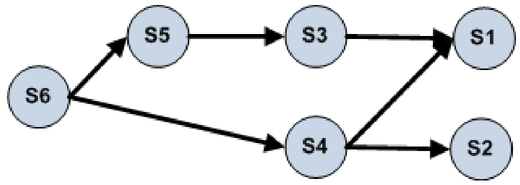

Figure 5: Sample Solution Digraph

Table 1: Propagation Index Updates

Solution	**Total** (γ_s)	$\Delta\gamma_s$
S1	2.50	0.5
S2	0.75	0.25
S3	1.50	0.5
S4	0.50	0.5
S5	0.5	0.5
S6	0	

3.4.2. Solution similarity

There are occasions when an agent wishes to find similar, or very different, solutions in the population. One metric that is often used in the literature to do this is the so-called Hamming distance, which is the number of binary variables with different values in two solutions. Although it can be useful in some contexts, that measure can be misleading for mixed-integer linear models. In most decision problems, some decisions have more importance than others. Often, a group of binary or integer variables is larger but of less significance. In the location-routing example discussed previously, many more decision variables are associated with the vehicle routing decisions than the location decisions, despite the fact that location decisions have a more lasting impact on the quality of the solution. For this problem, two solutions could have the exact same depot locations but have a high Hamming distance nonetheless, which would not reflect the importance of location decisions adequately. The same kind of drawback also occurs for other problems, like the graph coloring problem (Galinier, Hertz and Zufferey 2008).

In order to obtain a more accurate distance metric, one can measure the percentage of variables of each type that have the same value. Different types of variables can even be weighted in order to account for their relative importance. Suppose we have *A* types of integer variables in the optimization model being solved. Let $\mathbf{x}_a = (x_{an})_{n \in N_a}$ be the vector of variables of type $a = 1, ..., A$, N_a the index set of the variables x_{an} in $\mathbf{x}_a$, and λ_a the weighting factor associated with type a. Now consider two solutions $\mathbf{x}^{\mathrm{i}} = (\mathbf{x}_a^{\mathrm{i}})_{a=1,...,A}$ and $\mathbf{x}^{\mathrm{j}} = (\mathbf{x}_a^{\mathrm{j}})_{a=1,...,A}$ to the optimization model. The total weighted distance between these two solutions can be computed with the following formula:

$$D(\mathbf{x}^{\mathrm{i}}, \mathbf{x}^{\mathrm{j}}) = \sum_{a=1}^{A} \sum_{n \in N_a} \lambda_a \frac{\left| x_{an}^{\mathrm{i}} - x_{an}^{\mathrm{j}} \right|}{\left| N_a \right|}$$

4. EXPERIMENTAL TEST CASE

This section provides an experimental test case to illustrate how CAT can be applied to solve a specific complex decision problem, and to demonstrate the validity and the effectiveness of the approach.

More specifically, we show how to use CAT to solve multi-period Supply Chain Network (SCN) design problems.

4.1. Multi-Period Supply Chain Network Design Model

According to Santoso et al. (2005), SCN design is a crucial component of the planning activities of today's world-class manufacturing firms. As Klibi, Martel and Guitouni (2010) point out, typical SCN design problems involves strategic decisions on the number, location, capacity and missions of the production-distribution facilities a company should use to provide goods to a set of potential product-markets. A large number of particular formulations and solution methods have been proposed for single and multi-period versions of the SCN design problem, such as Vidal and Goetschalckx (2001), Paquet, Martel and Desaulniers (2004) and Martel (2005); the latter also provides an extensive overview of the relevant literature.

SCN design decisions are revised at the beginning of a set N of reengineering cycles each covering one or several planning periods $t \in T_n, n \in N$. Collectively, these reengineering cycles define the planning horizon $T = \cup_{n \in N} T_n$. Under the assumption that the future is known with certainty, the structure of the mathematical programming model to solve on a rolling horizon basis, for the prevailing SCN design paradigm, can be synthesized as follows. The following notation is used.

$\mathbf{x}_n^{\mathrm{p}}$: Vector of binary decision variables equal to 1 when using a given facility platform on a network site during reengineering cycle $n \in N$.

$\mathbf{x}_n^{\mathrm{s}}$: Vector of binary decision variables equal to 1 when a given sourcing/transportation contract is selected for reengineering cycle $n \in N$.

$\mathbf{x}_n^{\mathrm{d}}$: Vector of binary decision variables equal to 1 when a given demand shaping offer is selected for a product-market during reengineering cycle $n \in N$.

$\mathbf{x}_n = (\mathbf{x}_n^{\mathrm{p}}, \mathbf{x}_n^{\mathrm{s}}, \mathbf{x}_n^{\mathrm{d}})$: Vector of all binary design variables for cycle $n \in N$.

$\mathbf{y}_t^{\mathrm{f}}$: Vector of aggregate product flows on the network arcs in planning period $t \in T$.

$\mathbf{y}_t^{\mathrm{a}}$: Vector of aggregate activity levels (production or throughput) in the nodes (plants or depots) of the network in planning period $t \in T$.

$\mathbf{y}_t^{\mathrm{i}}$: Vector of aggregate inventory levels in the nodes of the network at the end of planning period $t \in T$.

$\mathbf{y}_t = (\mathbf{y}_t^{\mathrm{f}}, \mathbf{y}_t^{\mathrm{a}}, \mathbf{y}_t^{\mathrm{i}})$: Vector of all continuous activity level variables for period $t \in T$.

$\mathbf{p}_t^{\mathrm{f}}$: Vector of the unit prices paid for the delivery of the products associated with the flows $\mathbf{y}_t^{\mathrm{f}}$.

$\mathbf{c}_t$: Vector of the unit variable costs associated with the elements of activity vector $\mathbf{y}_t$.

$\mathbf{e}_t$: Vector of the capital recovery or contract expenditures in planning period *t* for the elements of design vector $\mathbf{x}_{n(t)}$ (*n*(*t*) denotes the cycle *n* including planning period *t*).

EVA_t : Economic value added by the SCN in planning period $t \in T$.

α : Discount rate used by the company, based on its weighted average cost of capital (WACC).

$\mathbf{b}_t^{\mathrm{c}}$: Vector of the *capacity* provided in period $t \in T$ by the platform, sourcing and transportation resources/contracts associated with the elements of design vector $(\mathbf{x}_{n(t)}^{\mathrm{p}}, \mathbf{x}_{n(t)}^{\mathrm{s}})$.

$\mathbf{b}_t^{\mathrm{d}}$: Vector of the potential *demand* available in period $t \in T$ under the demand-shaping offer associated with the elements of design vector $\mathbf{x}_{n(t)}^{d}$.

$\mathbf{b}_t = (\mathbf{b}_t^{\mathrm{c}}, \mathbf{b}_t^{\mathrm{d}})$: Vector of all capability parameter values associated with design vector $\mathbf{x}_{n(t)}$ for period $t \in T$.

In these definitions, *platforms* refer to alternative facility resource configurations that can be implemented on a site. They are characterized by technology and capacity choices to support a set of activities, and they involve specified capital recovery expenditures. A site without platform is not utilized. The platform on a site can change at the beginning of reengineering cycles to reflect opening, closing, expansion or reorganization decisions. Platforms are also used to characterize the proposals of potential subcontractors or public warehouses. In some formulations, complementary change-of-state binary variables are defined to facilitate the modeling of implementation expenses. Sourcing and transportation *contracts* specify prices and capacity for raw material and transportation service vendors. Demand-shaping *offers* are potential product-market selling policies specified in terms of price, response time, fill rate, or other order winning criterion. They influence demand and they may impose constraints on the network structure. Individual products daily/weekly procurement, production, inventory and shipping decisions are aggregated into flow, activity level and inventory level decisions for product families, demand zones and planning periods.

Using this notation, a typical multi-period SCN design model can be formulated as follows:

$$\max \sum_{t \in T} \frac{EVA_t}{(1-\alpha)^t}, \quad EVA_t = [\mathbf{p}_t^{\mathrm{f}} \mathbf{y}_t^{\mathrm{f}} - \mathbf{c}_t \mathbf{y}_t - \mathbf{e}_t \mathbf{x}_{n(t)}] \qquad (1)$$

subject to

$$\mathbf{V}_{n-1}\mathbf{x}_{n-1}^{\mathrm{p}} + \mathbf{W}_n \mathbf{x}_n^{\mathrm{p}} = \mathbf{u}_n \qquad n \in N \qquad (2)$$

$$\mathbf{x}_n \in X_n \qquad n \in N \qquad (3)$$

$$\mathbf{A}_t \mathbf{y}_t \le \mathbf{b}_t \mathbf{x}_{n(t)} \qquad t \in T \qquad (4)$$

$$(\mathbf{y}_{t-1}, \mathbf{y}_t) \in Y_t \qquad t \in T \qquad (5)$$

where $\mathbf{V}_{n-1}$, $\mathbf{W}_n$ and $\mathbf{A}_t$ are parameter matrices, $\mathbf{u}_n$ is a parameter vector, X_n is the set of feasible designs specified by local cycle *n* constraints, and Y_t is an activity level feasibility set for planning period *t*. The objective function (1) maximizes value creation over the planning horizon. Constraints (2) ensure that platforms are changed coherently from a reengineering cycle ($\mathbf{x}_{n-1}^{\mathrm{p}}$) to the next ($\mathbf{x}_{n}^{\mathrm{p}}$). Constraints (3) include additional cycle dependent constraints required to make sure that design options are properly selected. For example, during a cycle, one cannot operate more than one platform on a site or select more than one demand shaping offer for a product-market. Constraints (4) specify the activity level restrictions imposed in period $t \in T$ by the capabilities provided by the selected design $\mathbf{x}_{n(t)}$. These are mainly production-warehousing capacity constraints on $\mathbf{y}_{t}^{\mathrm{a}}$, storage capacity constraints on $\mathbf{y}_{t}^{\mathrm{i}}$, as well as vendor capacity, transportation capacity and potential demand constraints on $\mathbf{y}_{t}^{\mathrm{f}}$. Finally, (5) includes mainly flow conservation constraints on $\mathbf{y}_{t}$, and accounting constraints to calculate end-of-period inventories $\mathbf{y}_{t}^{\mathrm{i}}$ from initial inventories $\mathbf{y}_{t-1}^{\mathrm{i}}$ and relevant inflows/outflows in $(\mathbf{y}_{t}^{\mathrm{f}}, \mathbf{y}_{t}^{\mathrm{a}})$.

This streamlined formulation captures the main elements of multi-cycles SCN design models such as those proposed by Martel (2005), Thanh, Péton and Boste (2010), and Carle, Martel and Zufferey (2012). The model provided above is general and conceptual; a particular implementation may need some extra constraints imposing operational limits, such as maximum number of facilities to be built or expanded per planning cycle. The exact problem formulation used for the experimental tests, along with a detailed discussion of modeling choices, is provided in Carle, Martel and Zufferey (2012).

4.2. CAT Implementation

In the SCN design problem examined, three dimensional views were adopted to identify sub-problems and formulate associated sub-models. Each one is closely connected to the nature of the decision problem to be solved and to its mathematical formulation. Figure 6 presents these dimensional views, as well as the set of models, sub-models, and optimization algorithms used to tackle the SCN design problem.

4.2.1. Views, models and sub-models

At the integrated view level in Figure 6, the *mixed-integer optimization model* refers to the "complete model" presented in section 5.1. It is used as a basis for the creation of dimensional and integration sub-models. The *uncapacitated SCN design model* is a relaxation of the complete model in which vendors, transportation options and facilities are assumed to have infinite capacity. The *single-sourcing SCN design model* is a restricted version of the original model imposing products delivery to a given demand zone from a single source. This model is hard to solve but, since it replaces product flows by binary origin-destination-transportation assignment variables, a neighborhood-based local search algorithm can be elaborated to solve it.

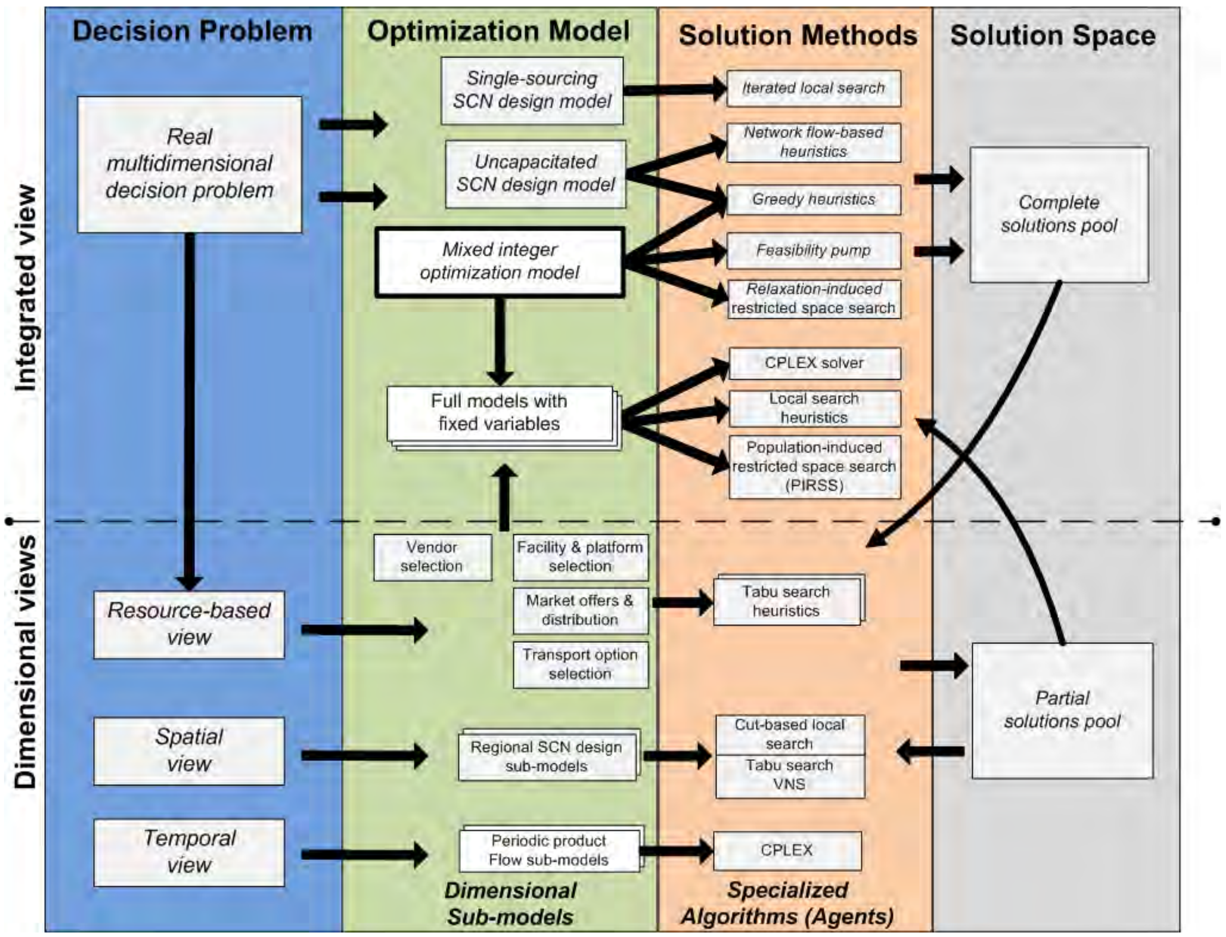
Decision Problem
Optimization Model
Solution Methods
Solution Space
Integrated view
Dimensional views
Real multidimensional decision problem
Single-sourcing SCN design model
Uncapacitated SCN design model
Mixed integer optimization model
Full models with fixed variables
Iterated local search
Network flow-based heuristics
Greedy heuristics
Feasibility pump
Relaxation-induced restricted space search
CPLEX solver
Local search heuristics
Population-induced restricted space search (PIRSS)
Complete solutions pool
Vendor selection
Facility & platform selection
Market offers & distribution
Transport option selection
Resource-based view
Tabu search heuristics
Partial solutions pool
Spatial view
Regional SCN design sub-models
Cut-based local search
Tabu search
VNS
Temporal view
Periodic product Flow sub-models
CPLEX
Dimensional Sub-models
Specialized Algorithms (Agents)

Figure 6: CAT Constructs for the Multi-Period SCN Design Problem

The resource-based view refers to the main types of internal (production and distribution center's platforms) and external (vendors, carriers and customers) resources of the SCN. In a business context, these resources tend to be managed by different responsibility centers; it is then intuitive to formulate sub-models that match the decision sub-problem faced by each of these centers. Table 2 presents the continuous and binary decision variables associated with the sub-models thus obtained. The integration of partial solutions along the resource-based view requires a linking integration sub-model that is a variant of the classical transportation model, and this LP is rather easy to solve. In order to decrease the risk of infeasibility in the integration phase, additional valid inequalities are added into each dimensional sub-model, stating that a minimum level of capacity is needed in order to meet the network's expected total customer demand for each product. Another source of infeasibility is insufficient transportation capacity, especially when connecting vendors to facilities. A heuristic that selects one additional transportation option to be included in the integration sub-model was designed to circumvent this problem.

The spatial view refers to the geographical positioning of business entities such as sales territories, national divisions or subsidiaries. The SCN design problem lends itself particularly well to spatial partitioning, since several companies split the logistics responsibilities into national or territorial divisions. Two strategies are used for spatial decomposition:

- Sales territories are used to form non-overlapping sub-models; each facility, vendor and demand zone is located within exactly one sub-model. If no vendor is able to supply a given product in a territory, it is assumed that the product will sent from another sales territory.
- The whole geographical area is divided into, typically, 5 or 6 sub-models according to proximity between locations. A facility is selected at random from all potential facilities, and then a virtual sales territory is constructed by adding the *Y* nearest vendors, facilities, and demand zones.

Sub-model:	Sourcing	Facility location and configuration	Demand shaping and distribution	Transportation options selection	Integration sub-model
Binary decision variables	• Vendor selection	• Facility location • Platform selection	• Distribution center location • Distribution center platform selection • Market offer selection	• Transportation option selection	
Continuous decision variables	• Product quantity purchased per period	• Product flows between facilities • Facility throughputs • Inventories	• Product flows between warehouses and demand zones • Facility throughputs in warehouses • Inventories	• Product flows by transportation option	• Flows between vendors and facilities • Flows between facilities and demand zones • Carry-over inventories

Table 2: Sub-Models Associated with the Resource-Based View

Typically, the spatial view integration sub-models fix the values of product flows between nodes (vendors, facilities and demand zones) located in different territory. Spatial partitioning is very different than resource-based partitioning since each sub-model is essentially a much smaller version of the original problem. Decisions variables of each type are present in each sub-model. Furthermore, spatial sub-models offer different advantages and challenges than resource-based sub-models. It is possible to design merging integration sub-models by merging solutions from territories specified with different partitioning strategies in order to expand the search space.

The temporal view refers to the fact that SCN design problems must be solved over a long multi-year planning horizon. Although only the decisions from the first period are typically implemented, one wants to anticipate future needs and challenges. As explained earlier, these problems consider two time frames: reengineering cycles and the annual time periods. Since SCN design decisions are made at the reengineering cycle level rather than at the period level, it makes sense to define sub-problems by partitioning the planning horizon into cycles. In our case study, the complete model was thus partitioned into the three sub-models described in Table 3. For this view, the only linking variables in the integration sub-model are inventory carry-overs between time periods from different sub-models. Even when using concave piecewise-linear inventory-throughputs functions, the integration sub-model remains solvable in a few seconds of computation time with CPLEX.

Table 3: Time-Based Partitioning

Sub-model (cycle $n \in N$)	Planning period $t \in T_n$
1	$\{1,2\}$
2	$\{3,4,5\}$
3	$\{6,...,10\}$

4.2.2. Agents and algorithms

The CAT system developed to solve the multi-period activity-based SCN design problem has 16 types of agents. The vast majority of the agents work on sub-models or on restricted/relaxed versions of the complete optimization model. Table 4 presents the most important features of each agent: its name, type, the number of different solution methods it implements, whether the agent has a full (F) or partial (P) dimensional view, as well as the models they focus on, if any. Agents marked with an asterisk (*) in the *Method* column use version 12.1 of IBM ILOG CPLEX® in one or more of their optimization algorithms, either to solve sub-models or relaxed/restricted complete models to optimality, or as a heuristic by ending the solving process prematurely. Since CAT uses more than 40 different heuristics, it is not possible to provide the pseudo-code for each of them in the paper. A general outlook of the methods of each agent, along with references to similar heuristics, is provided in Carle, Martel and Zufferey (2012). All heuristics and agents are coded in C# and VB.NET 2008, and each agent is an executable program.

More than half of the agents (9/16) are improvement agents. The SCN design problem is complex and several sub-models are quite challenging. Each improvement agent is designed to solve a specific model or sub-model. ILS works on severe restrictions of the complete model, under the assumption that, for a given product, a facility or demand zone is supplied by at most one source. CBLS is a local search algorithm that removes the variables that are most prominent in the solutions population from the search space. Although the solutions they yield are not of exceptional quality, their purpose is to find solutions that are radically different than those already in the population, in order to prevent a premature convergence of the algorithm. This principle is known as *diversification* in the metaheuristics literature (Gendreau and Potvin 2010).

Agent	Type	Method	Functional	Spatial	Time	Models
FPump	C	6*	F	F	F	Complete model
Greedy	C	8	F	F	F	Complete model and Uncapacitated SCN design model
RIRSS	C	2*	F	F	F	Complete model
BasicNet	C	3	P	P	F	Facility location, network flow, transportation model
TSV	I	1	P	F	F	Sourcing sub-model
TSI	I	2	P	F	F	Facility location and configuration sub-model
TSD	I	1	P	F	F	Demand shaping and distribution sub-model
TransOpt	I	1	P	F	F	Transportation option sub-model
RegionalTS	I	2	F	P	F	Regional SCN design sub-model
CBLS	I	2*	P	P	P	Regional SCN design sub-model
FlowOpt	I	2*	F	F	P	Periodic network flow, time dimension linking model
CPLEX-SP	I	1*	F	F	F	Complete model
ILS	I	1	F	F	F	Complete model
Terminator	D	3	F	F	F	None; archives bad solutions
Integrate	T	7*	F	F	F	Integration sub-models
PIRSS	T	2*	F	F	F	Integration sub-models

Agent types: C – Construction; I – Improvement; D – Destruction; T – Integration.

Table 4: CAT Agents Implemented to Solve the SCN Design Problem

The two integration agents use different search strategies. The *Integrate* agent solves integration sub-models aiming at fixing the values of linking variables. Some of the linking sub-models are solved to optimality using CPLEX while others are treated through the use of heuristics. Since the improvement agents produce a large quantity of partial solutions, the speed of the integration process is of prime importance. As such, it would be impractical, both in terms of computation time and RAM, to solve each integration sub-problem to optimality. In contrast, the PIRSS agent focuses on merging integration sub-models. It effectively models the solution space formed by the union of two or more solutions as a restricted MIP, and then explores it thoroughly within a time limit using CPLEX. For instance, partial solutions from the sub-models of the resource-based dimension are merged with a partial solution from a spatial sub-model.

In our implementation, the destruction agent starts to remove solutions from the population when it reaches 50 complete solutions, or 15 partial solutions to any sub-model.

5. COMPUTATIONAL RESULTS

In order to validate and assess the CAT problem solving approach, a set of 25 benchmark problem instances were generated. The first 20 of these instances (labeled B-01 to B-20) are based on a realistic case representing a typical B2B company manufacturing and selling products through the United States. Product demands and prices, transportation costs as well as the fixed and variable costs of each platform, vendor offer and transportation options are randomly generated, but are based on realistic parameter value ranges found in Ballou (1992). The remaining 5 (labeled G-01 to G-05) are based on randomized values using a procedure similar to the one described in Cordeau, Pasin and Solomon (2006) to generate low-capacity, high flow magnitude instances.

The potential supply chain network comprises 9 to 18 potential production-distribution facilities, 30 to 60 potential distribution centers, 100 to 300 demand zones representing clusters of customers in the vicinity of major U.S. cities, and 50 to 300 vendor offers. For the production-distribution facilities, 3 to 8 alternative platforms are considered, and up to 4 capacity expansion upgrades are available for each of these platforms. For the distribution centers, 5 alternative platforms are considered, with a maximum of 2 upgrades per platform. Five to 16 product families are sold to customers, and their bill-of-materials include 10 to 60 components. Several transport capacity options are modeled; truckload and less-than-truckload shipping are considered, both in the form of a limited-size private fleet and long-term truck leasing. A common carrier with a large capacity is also available. Five demand shaping offers are considered for each product. All problem instances involve concave inventory-throughput functions. When the models are solved with CPLEX, these concave functions are approximated by 3-segments piecewise linear functions using a procedure similar to the one described in Amrani et al. (2011). The resulting complete models each have millions of continuous variables as well as from 20,000 to 200,000 binary variables. Note that this problem set is different than the one used to validate the CAT system. This guarantees that the system is not custom-built or fine-tuned for a given set of instances.

In order to assess the impact of the main CAT components on overall performance, three versions of CAT were tested. The list of individual agents included in each version is provided in

Table 5. The “greedy” version contains the agents whose mission is to quickly generate several solutions for the other agents to improve upon. The “basic version” contains all the agents found in the greedy version as well as those built to tackle the dimensional sub-models, but it lacks any sophisticated integration agent. The complete version contains the 16 agents listed in Table 4.

Version	Agents included
Greedy	FPump, Greedy, BasicNet, Terminator
Basic	FPump, Greedy, BasicNet, Terminator, TSV, TSI, TSD, TransOpt, RegionalTS, FlowOpt and CPLEX-SP
Complete	All agents listed in Table 4

Table 5: Agents Included in each Version of CAT

All the experiments were performed on a dual 2.66 GHz 64-bit Intel Xeon® computer with 64 GB of RAM. Both CPLEX and CAT were allowed to use the twelve available processors as needed. Table 6 presents the computational results for our 25 benchmark instances with a time limit of 60 minutes. For each solution method (CAT-Greedy, CAT-Basic, CAT-Complete, and CPLEX), 10 runs were executed for every problem instance. For each run on each instance, the best feasible solution found by the method (*BSol*) is recorded. For a run, the gap between *BSol* and the best solution found for a problem instance over all runs and solution methods *(BSol*)* is then computed as $100 \times \left| BSol^* - BSol \right| / \left| BSol^* \right|$. *Avg* indicates the average gap obtained over 10 runs for a method, and *Best* the best gap among these 10 runs.

		Greedy		CAT-Basic		CAT-Complete		CPLEX	
Instance	*(BSol*)*	Best	Avg	Best	Avg	Best	Avg	Best	Avg
B-01	238798247	52.71%	59.93%	11.73%	18.24%	5.39%	7.86%	13.68%	15.37%
B-02	249367184	56.77%	65.47%	13.40%	20.54%	3.88%	6.80%	17.52%	19.25%
B-03	267302131	63.28%	75.02%	18.66%	24.84%	5.78%	7.50%	22.40%	25.95%
B-04	269632962	61.53%	76.14%	19.51%	24.76%	2.62%	6.89%	21.43%	24.02%
B-05	278667490	62.65%	76.19%	14.73%	19.47%	4.36%	6.95%	19.60%	19.84%
B-06	276510121	68.08%	79.02%	12.02%	17.50%	5.98%	8.53%	16.29%	18.26%
B-07	356662807	70.37%	84.12%	11.85%	18.03%	7.34%	9.48%	13.28%	14.37%
B-08	422735400	76.75%	86.07%	12.55%	18.88%	5.85%	8.99%	14.17%	18.60%
B-09	475093862	80.83%	87.83%	18.17%	25.54%	5.61%	7.89%	21.92%	26.81%
B-10	503154315	79.31%	86.29%	16.61%	22.67%	2.29%	4.96%	25.86%	27.75%
B-11	551335586	81.71%	87.95%	11.35%	16.31%	6.32%	9.80%	23.73%	25.54%
B-12	508660093	81.15%	87.97%	19.45%	24.70%	8.39%	10.96%	19.39%	19.82%
B-13	518215224	81.31%	87.58%	15.34%	21.82%	8.83%	10.11%	22.83%	24.75%
B-14	551371409	84.69%	90.38%	13.64%	19.49%	7.23%	9.81%	19.08%	20.73%
B-15	511565563	79.91%	84.37%	10.58%	15.92%	2.42%	5.60%	23.45%	24.47%
B-16	560355024	83.89%	90.54%	12.99%	19.61%	10.61%	13.27%	30.51%	31.00%
B-17	592086814	84.00%	90.16%	14.36%	21.44%	8.71%	10.72%	28.02%	30.96%
B-18	597498896	85.59%	91.01%	11.38%	18.01%	7.82%	10.01%	26.27%	28.52%
B-19	541063880	81.62%	91.11%	14.69%	22.39%	7.22%	9.66%	22.21%	25.37%
B-20	583337610	82.08%	86.76%	13.88%	19.78%	5.67%	8.19%	22.44%	23.63%
G-01	341707025	69.80%	80.29%	11.79%	17.95%	6.99%	9.45%	21.35%	21.98%
G-02	412869182	74.83%	85.72%	18.21%	22.81%	4.60%	7.04%	20.48%	21.96%
G-03	773059691	86.78%	90.77%	10.10%	15.46%	10.44%	12.46%	19.23%	22.99%
G-04	191659172	92.63%	98.69%	13.94%	22.02%	3.87%	9.46%	26.75%	31.40%
G-05	532929990	80.71%	86.54%	13.82%	20.80%	5.96%	9.16%	22.15%	23.07%
Average:		76.12%	84.24%	14.19%	20.36%	6.17%	8.86%	21.36%	23.46%

Table 6: Computational Results after 60 Minutes

As noted earlier, it is rather obvious that the only benefit of using a greedy strategy is that it provides solutions for the improvement agents to work on. All the agents in the Greedy implementation typically complete their work in less than 10 minutes of computation time. By adding improvement agents, the average gap over all instances drops from 84.24% to 20.36%. CAT-Basic's average performance (20.36%) is slightly better than CPLEX's (23.46%). However, CAT-Complete clearly outperform both CPLEX and CAT-Basic, with an average gap of 8.86%. On average, CAT-Complete outperforms CAT-Basic by 11.50% and CPLEX by 14.59%. One may notice that the gaps shown here are fairly high; since the SCN design problem maximizes net profits (Revenues - Costs), the objective function represents a relatively small percentage of the company's actual revenues (between 3 to 9% for our instances). Consequently, for a net profit of 3%, a 1% reduction in costs provides an increase in objective function profits of about 25%.

		Greedy		CAT-Basic		CAT-Complete		CPLEX	
	(BSol)*	Best	Avg	Best	Avg	Best	Avg	Best	Avg
B-101	238798247	52.71%	59.93%	1.37%	2.91%	0.00%	0.58%	5.16%	5.94%
B-102	249367184	56.77%	65.47%	6.79%	8.93%	0.00%	0.66%	10.72%	12.91%
B-103	267302131.2	63.28%	75.02%	6.98%	9.43%	0.00%	0.88%	11.74%	13.71%
B-104	269632961.9	61.53%	76.14%	2.27%	5.28%	0.00%	1.05%	10.01%	11.85%
B-105	278667489.8	62.65%	76.19%	7.81%	9.32%	0.00%	1.54%	14.11%	15.36%
B-106	276510120.5	68.08%	79.02%	5.97%	7.77%	0.00%	1.79%	11.84%	13.23%
B-107	356662806.7	70.37%	84.12%	5.42%	7.51%	0.00%	2.18%	10.46%	10.67%
B-108	422735400.5	76.75%	86.07%	6.89%	8.29%	0.00%	2.16%	9.55%	10.86%
B-109	475093862.1	80.83%	87.83%	11.61%	13.94%	0.00%	1.96%	11.56%	14.68%
B-110	503154315	79.31%	86.29%	9.88%	11.97%	0.00%	2.01%	12.35%	13.15%
B-111	551335585.9	81.71%	87.95%	5.47%	7.73%	0.00%	1.63%	11.07%	14.11%
B-112	508660093.1	81.15%	87.97%	12.68%	14.81%	0.00%	0.53%	10.50%	12.15%
B-113	518215224.3	81.31%	87.58%	7.85%	9.55%	0.00%	1.87%	11.68%	12.97%
B-114	551371408.6	84.69%	90.38%	6.74%	8.32%	0.00%	1.88%	11.66%	12.60%
B-115	511565563.3	79.91%	84.37%	4.26%	6.61%	0.00%	0.46%	10.13%	12.92%
B-116	560355023.8	83.89%	90.54%	6.75%	8.62%	0.00%	1.85%	17.24%	18.58%
B-117	592086813.7	84.00%	90.16%	8.27%	10.21%	0.00%	1.36%	12.29%	14.11%
B-118	597498895.9	85.59%	91.01%	5.24%	6.50%	0.00%	0.58%	9.52%	11.33%
B-119	541063880.3	81.62%	91.11%	7.64%	9.62%	0.00%	1.46%	13.46%	13.87%
B-120	583337609.8	82.08%	86.76%	5.34%	8.18%	0.00%	1.87%	12.25%	12.60%
G-1	341707025.3	69.80%	80.29%	5.40%	8.14%	0.00%	0.78%	11.69%	13.71%
G-2	412869181.8	74.83%	85.72%	12.69%	14.61%	0.00%	1.21%	12.77%	13.56%
G-3	773059691.1	86.78%	90.77%	7.73%	10.04%	0.00%	1.30%	12.02%	12.82%
G-4	191659172.2	92.63%	98.69%	7.63%	10.36%	0.00%	2.51%	16.61%	18.46%
G-5	532929989.6	80.71%	86.54%	5.49%	8.35%	0.00%	1.74%	12.04%	13.99%
Average:		76.12%	84.24%	6.97%	9.08%	0.00%	1.43%	11.70%	13.21%

Table 7: Computational Results after 10 Hours (600 Minutes)

Table 7 presents computational results for the same 25 instances after 600 minutes (10 hours) of computation time. The Greedy approach's performance is unchanged, as it needs only a few minutes to complete. CAT-Basic's average gap over all instances drops from 20.36% to 9.08%, a 11.28% decrease, while CAT-Complete's average gap drops from 8.86% to 1.43%, a 7.43% decrease. CPLEX's gap decreases by 10.25%, from 23.46% to 13.21%. These results clearly show the value added by diversification and the payoff of integration agents: on average across all instances, CAT-Complete's performance beats CPLEX by 11.77% and CAT-Basic by 7.65%. Furthermore, it finds the best known solution on all 25 instances.

It is interesting to note that although agents of the proposed CAT system use some acceleration techniques to reduce the time required to perform read/write operations, the system itself has been built as a prototype. For instance, agents are not allowed to share memory structures or solution components directly through the computer's memory even if they work on the same machine; each agent reads information from the blackboard and writes to the blackboard, resulting in a lot of non-optimal hard disk activity. In a production or commercial version, additional implementation improvements would reduce computational times substantially.

6. CONCLUSION

This paper proposes a generic approach to model and solve complex real-world decision problems. It shows how to look at decision problems from different point-of-views and how to partition the problem, and the associated optimization model, into dimensional sub-models. It also proposes CAT, a new agent-based metaheuristic designed to benefit from the complexity reductions resulting from the multi-dimensional views of the problem. This metaheuristic is very scalable since its execution can easily be distributed over multiple computers. Some general implementation guidelines to design CAT systems were proposed, and an experimental case study involving a supply chain network design problem was presented. Experimental results clearly showed the benefits of using the partitioning and integration mechanisms presented earlier.

Given the decreasing costs of multi-core processors and physical memory, parallel and distributed optimization strategies have become much more practical to tackle large-scale decision problems. CAT is easily extendable by adding new agents or processing power as needed or by allowing some of the agents to work using more than one processor at a time. More precise guidelines to assess the impact and utility of a given agent could and should be elaborated, in order to limit the number of agents used.

An obvious opportunity for further research is to apply CAT to other optimization models. The location-routing and production-inventory-routing models seem promising, as are complex production scheduling models. CAT could also be extended to model decision problems with multiple decision makers, in cooperative or non-cooperative contexts. Another straightforward extension would be to address multi-firm supply chain network design problems, where sub-problems could be designed for the set of facilities owned by each participating firm. The agent structure would enable the use of private information in the optimization sub-models without the need to share such information with supply chain partners.

7. REFERENCES

Achterberg, T., T. Berthold. 2007. Improving the Feasibility Pump. *Discrete Optimization*, 4, 77-86.

Amrani, H., A. Martel, N. Zufferey, P. Makeeva. 2011. A variable neighborhood search heuristic for the design of multicommodity production-distribution networks with alternative facility configuration. *OR Spectrum*, 33(4), 989-1007.

Aydin, M. E., T. C. Fogarty. 2004. Teams of autonomous agents for job-shop scheduling problems: An experimental study. *Journal of Intelligent Manufacturing*, 15, 455-462.

Balinski, M. 1961. Fixed-cost Transportation Problems. *Naval Research Logistics Quarterly*, 8, 41-54.

Ball, M. O. 2011. Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science,* 16, 21-38.

Ballou, R. H. 1992. *Business Logistics Management*, 3rd edition, Prentice Hall.

Beasley, J. E. 1993. Lagrangean heuristics for location problems. *European Journal of Operational Research*, 65, 383-399.

Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4, 238-252.

Bixby, R. E., E. Rothberg, Z. Gu. 2010. The lastest Advances in Mixed-Integer Programming Solvers, presentation notes from the CIRRELT Spring School on Combinatorial Optimization in Logistics, Montreal.

Blum, C., J. Puchinger, G. R. Raidl, A. Roli. 2011. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11, 4135-4151.

Carle, M.-A., A. Martel, N. Zufferey. 2012. The CAT Metaheuristic for the Solution of Multi-Period Activity-Based Supply Chain Network Design Problems. *International Journal of Production Economics*, *In Press,* http://dx.doi.org/10.1016/j.ijpe.2012.06.016.

Cordeau, J.-F., G. Laporte. 2005. Tabu Search Heuristics for the Vehicle Routing Problem, In Sharda, Voss, Rego and Alidaee (Eds), *Metaheuristic Optimization Via Memory and Evolution*, Operations Research / Computer Science Interfaces Series, Springer, Germany, 145-163.

Cordeau, J.-F., G. Laporte. 2007. The dial-a-ride problem: models and algorithms, *Annals of Operations Research*, 153(1), 29-46.

Cordeau, J.-F., F. Pasin, M. M. Solomon. 2006. An integrated model for logistics network design. *Annals of Operations Research*, 144(1), 59-82.

Crainic, T. G., Y. Li, M. Toulouse. 2006. A First Multilevel Cooperative Algorithm for Capacitated Multicommodity Network Design. *Computers & Operations Research*, 33(9), 2602–2622.

Danna, E., E. Rothberg, C. LePape. 2005. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102, 71-90.

Dantzig, G. B., P. Wolfe. 1960. Decomposition Principle for Linear Programs. *Operations research*, 8, 101-111.

Feo, T. A., M. G. C. Resende. 1989. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8, 67-71.

Fischetti, M., A. Lodi. 2003. Local Branching. *Mathematical Programming,* 98(1), 23-47.

Galinier, P., A. Hertz, N. Zufferey. 2008. An Adaptive Memory Algorithm for the Graph Coloring Problem. *Discrete Applied Mathematics*, 156 (2), 267 – 279.

Gaudreault, J., J.-M. Frayret, G. Pesant. 2009. Distributed Search for Supply Chain Coordination. *Computers in Industry*, 60(6), 441-451.

Gendreau, M., J.-Y. Potvin. 2010. Tabu Search. In M. Gendreau and J.-Y. Potvin (Eds), *Handbook of Metaheuristics*, 2nd Edition, Springer-Verlag Berlin, Germany, 41-60.

Gorder, P. F. 2007. Multicore processors for science and engineering. *Computing in Science & Engineering*.

Hertz, A., M. Plumettaz, N. Zufferey. 2008. Variable Space Search for Graph Coloring. *Discrete Applied Mathematics*, 156(13), 2551-2560.

Keskinocak, P., F. Wu, R. Goodwin, S. Murthy, R. Akkiraju, S. Kumaran, A. Derebail. 2002. Scheduling solutions for the paper industry. *Operations Research*, 50(2), 249-259.

Klibi, W., A. Martel, A. Guitouni. 2010. "The design of robust value-creating supply chain networks: A critical review." *European Journal of Operational Research* **203**: 283-293.

Klose, A., A. Drexl. 2005. Facility location models for distribution system design. *European Journal of Operational Research*, 162, 4-29.

Maniezzo, V., T. Stützle, S. Voss. 2009. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, Springer.

Martel, A. 2005. The Design of Production-Distribution Networks: A Mathematical Programming Approach. In Geunes and Pardalos (Eds), *Supply Chain Optimization*, 265-306.

Melab, N., E.-G. Talbi, S. Cahon, E. Alba, G. Luque. 2006. Parallel Metaheuristics: Models and Frameworks. in Talbi, E.-G. (Ed.), *Parallel Combinatorial Optimization*, Wiley.

Melo, M. T., S. Nickel, F. Saldanha-da-Gama. 2012. An efficient heuristic approach for a multi-period logistics network redesign problem, *TOP*, *In Press*, DOI 10.1007/s11750-011-0235-3.

Meneses, C. N., P. M. Pardalos, M. Ragle. 2008. Asynchronous Teams for probe selection problems. *Discrete Optimization*, 5, 74-87.

Michel, L., P. Van Hentenryck. 2004. A simple Tabu search for warehouse location. *European Journal of Operational Research*, 157, 576-591.

Min, H., V. Jayaraman, R. Srivastava. 1998. Combined location-routing problems: A synthesis and future research directions. *European Journal of Operational Research*, 108, 1-15.

Moscato, P., C. Cotta. 2010. A Modern Introduction to Memetic Algorithms. In Gendreau, M. and Potvin, J.-Y. (Eds), *Handbook of Metaheuristics*, 2nd Edition, Springer, 141-183.

Murthy, S., R. Akkiraju, R. Goodwin, P. Keskinocak, J. Rachlin, F. Wu, J. Yeh, R. Fuhrer, S. Kumaran, A. Aggarwal, M. Sturzenbecker, R. Jayaraman, R. Daigle. 1999. Cooperative Multiobjective Decision Support for the Paper Industry. *Interfaces*, 29(5), 5-30.

Nagy, G., S. Salhi. 2007. Location-routing: issues, models and methods. *European Journal of Operational Research,* 177, 649-672.

Paquet, M., A. Martel, G. Desaulniers. 2004. Including technology selection decisions in manufacturing network design models. *International Journal of Computer Integrated Manufacturing*, 17(2), 117-125.

Raidl, G. R., J. Puchinger. 2008. Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization. In Blum C., Aguilera M.J.B., Roli A., Sampels, M. (Eds), *Hybrid Metaheuristics: An Emerging Approach to Optimization*, Springer, 31-62.

Raidl, G. R., J. Puchinger, C. Blum. 2010. Metaheuristic Hybrids. In Gendreau M. and Potvin J.Y. (Eds) *Handbook of Metaheuristics*, 2nd edition, Springer, 469-491.

Ratajczak-Ropel E. 2010. Experimental Evaluation of the A-Team Solving Instances of the RCPSP/max Problem. In Jedrzejowicz et al. (Eds): *KES-AMSTA 2010, Part II, LNAI 6071*, 210–219.

Rodney D., A. Soper, C. Walshaw. 2007. The Application of Multilevel Refinement to the Vehicle Routing Problem. In Fogel D. *et al.* (Eds), *Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling*, 212–219.

Santoso, C., S. Ahmed, M. Goetschalckx, A. Shapiro. 2005. A stochastic programming approach for supply chain network design under uncertainty. *European Journal of Operational Research,* 167, 96-115.

Schneeweiss, C. 2003. Distributed Decision Making, 2nd edition, Springer-Verlag Berlin, Germany.

Talbi, E.-G. 2002. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8, 541-564.

Talbi, E.-G. 2009. *Metaheuristics: From Design to Implementation*, Wiley.

Talukdar, S., S. Murthy, R. Akkiraju. 2003. Asynchronous teams. In Glover, F., Kochenberger, G.A. (Eds), *Handbook of Metaheuristics*, Kluwer Academic Publishers.

Thanh, P.N., O. Péton, N. Bostel. 2010. A linear relaxation-based heuristic approach for logistics network design. *Computers and Industrial Engineering*, 59, 964-975.

Toulouse, M., K. Thulasiraman, F. Glover. 1999. Multi-level Cooperative Search: A New Paradigm for Combinatorial Optimization and an Application to Graph Partitioning. In P. Amestoy *et al.* (Eds), Proc. Euro-Par'99 Parallel Processing, Lecture Notes in Computer Science, Vol. 1685. Berlin: Springer, 533–542.

Vidal, C., M. Goetschalckx. 2001. A global supply chain model with transfer pricing and transportation cost allocation. *European Journal of Operational Research,* 129, 134-158.

Walshaw, C. 2004. Multilevel Refinement for Combinatorial Optimization Problems. *Annals of Operations Research*, 131, 325-372.

Wu, T.-H., C. Low, J.-W. Bai. 2002. Heuristic solutions to multi-depot location-routing problems. *Computers and Operations Research,* 29, 1393-1515.

Zufferey, N. 2012. Metaheuristics: some Principles for an Efficient Design. *Computer Technology and Applications,* 3, 446-462.