International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 10 Issue: 11 | Nov 2023 www.irjet.net p-ISSN: 2395-0072

Impact of System Constraints on the Performance of Constraint Solvers in Optimizing the Schedules through Algorithm Choices

S.M.Suriyaarachchi, Wickramarathna W.A.Y.A, Senanayake H.R.U.D, Bandara D.M.T.D

***Abstract*— In the realm of combinatorial optimization, the efficiency and effectiveness of constraint solvers play a pivotal role in resolving complex scheduling problems. Solving these problems requires solver engines which require heavy computational power. Constraint solvers are a unique approach to solving these scheduling problems. This research delves into the intricate interplay between system constraints and the performance of constraint solvers when applied to the task of optimizing schedules, with a particular emphasis on the impact of algorithm choices. The primary objective of this study is to explore system-level limitations, including memory allocation, to enhance constraint solver performance and provide insights for strategies. Empirical investigation focuses on designing experiments showcasing system constraints on scheduling problems, analyzing sensitivity to constraints using various optimization solvers, and measuring solution quality, convergence rate, and resource consumption. This research reveals the dynamic interplay between system constraints and system-level limitations in combinatorial optimization, guiding practitioners and researchers in algorithmic choices, strategy adjustments, and resource allocations for complex scheduling problems.**

Keywords—Constraint Solvers, Optimizing Schedules, System Constraints, Scheduling problems.

I. INTRODUCTION

Constraint satisfaction has become a key paradigm in the fields of optimization and computational problem-solving for handling challenging real-world issues. The use of constraint solvers, specialized software tools made to locate workable solutions within the bounds of preset constraints, is essential in a variety of fields, including manufacturing, project management, artificial intelligence, and scheduling. Schedule optimization, particularly when combined with method selection, stands out as a fundamental issue in this field, as the interaction between system restrictions and solver performance assumes utmost significance [2].

Across many industries, it is crucial to allocate time, resources, and responsibilities effectively. The objective is always the same, regardless of whether the activity at hand is managing computing workloads in data centers or scheduling tasks in manufacturing plants or coordinating supply chain activities. Attaining optimal resource use while following a variety of limitations [3]. These problems can be approached methodically with the help of constraint solvers, which are powered by complex algorithms. However, as systems and tasks become more complicated, the restrictions placed on constraint solvers have a significant impact on their capacity to move through complex constraint landscapes.

The relationships between system restrictions and constraint solver performance in this situation call for careful investigation. System limitations cover a wide range of elements, such as the structure of the particular issue instance, available computational resources, memory accessibility, and processor capacity. As projects develop or new requests materialize, these restrictions, which are fundamentally dynamic, may change. Therefore, it is essential to understand how changes in these system restrictions affect how well constraint solvers perform in order to develop techniques that result in reliable and effective solutions [4].

An additional level of complexity is added by algorithm selection, a key component of this research. The solver's capacity to maneuver through the search space of potential solutions might be considerably impacted by the algorithm they use. Others may be better at handling complicated constraint relationships but require more processing power, while some algorithms may perform better in settings with constrained computational resources but fall short when presented with complex issue structures. Consequently, the interaction between system limitations, algorithm choice, and solver performance creates a multidimensional conundrum that merits careful examination.

In the context of optimizing schedules through algorithm selection, this study sets out on a quest to understand the complex relationship between system constraints and the effectiveness of constraint solvers [5].

The goal of this work is to examine this complex interplay in order to get insights that will not only advance theoretical knowledge but also provide practitioners with practical information for overcoming the difficulties presented by actual optimization scenarios. This study aims to add to the expanding corpus of theoretical understanding that supports successful constraint satisfaction and optimization strategies using a combination of empirical analysis, algorithmic investigation, and theoretical inquiry.

The following parts will examine the pertinent literature, lay out the theoretical underpinnings, explain the study methods, provide the results, and finally summarize the consequences of our inquiry [6]. With this extensive project, we hope to provide light on the complex dynamics regulating how system constraints affect constraint solver effectiveness, particularly when complex choices of algorithm are involved.

II. LITERATURE REVIEW

From manufacturing and transportation to project management and computer systems, scheduling optimization is a challenging problem that affects a wide range of businesses. Constraint solvers have become a powerful tool for navigating the complex landscape of constraints that are inherent in scheduling as a result of the major evolution of the computational approaches used to address these problems [7]. This study of the literature examines the state-of-the-art regarding the complex interaction between system constraints, algorithmic decisions, and the efficiency of constraint solvers when used to optimize schedules. This study examines previous studies in an effort to provide light on the complex interactions that affect how efficient and successful these solutions are.

2.1 Algorithm Choices and Solver Performance

In order to constraint solvers to be efficient and effective in tackling schedule optimization problems, the choice of suitable algorithms is crucial. The choice of algorithm is an important factor in the performance of solvers because different algorithms exhibit different strengths and weaknesses. Scheduling issues have been tackled by methods including constraint programming, mixed integer programming, and heuristic approaches, each of which targets a different part of the issues. For example, mixed integer programming excels at capturing complex dependencies inside scheduling issues, whereas constraint programming provides a flexible framework for modeling sophisticated constraints [8]. The choice of algorithm must be carefully considered based on the nature of the problem and the constraints imposed.

2.2 System Constraints and Solver Behavior

When used in scheduling optimization, system constraints cover a range of elements that affect how constraint solvers behave [1]. These limitations include the amount of computational capacity, the amount of memory, the processing speed, and the difficulty of the particular issue instance. The performance and behavior of the solver are greatly influenced by the availability of computational resources, according to research. When resources are limited, solvers adapt by using techniques like branch pruning or heuristic-guided search to reduce runtime [9]. Additionally, problem-specific restrictions like time frames and interaction between orders of precedence have a significant impact on solver efficiency. Research has looked on the dynamic adaptation of solver algorithms to various system constraints, enabling more efficient schedule optimization.

2.3 Domain-Specific Applications

The application of constraint solvers in optimizing schedules is not limited to a single domain but spans a wide range of applications [10]. For instance, constraint programming has been used to effectively manage task allocation in work task scheduling, where tasks need to be assigned to available resources while conforming to certain constraints. Similar to this, technician dispatching involves assigning technicians to service tasks depending on variables including location, skill level, and time limitations. In a variety of disciplines, researchers have looked into the use of constraint solvers to address these difficult problems.

2.4 Solver Performance Analysis

To evaluate the effectiveness of various constraint solvers in particular scheduling scenarios, numerous empirical studies have been carried out. Comparative analyses of mixed integer programming solvers SCIP and CBC against Optaplanner and MiniCP have revealed information about their effectiveness and applicability for various problem types. Researchers and practitioners can learn how different solvers perform under various system constraints by using these assessments, which often quantify characteristics like solution quality, runtime, and scalability. When choosing solvers and algorithms for particular schedule optimization tasks, these evaluations help provide the empirical groundwork for wise choices.

By focusing on these aspects, this work hopes to add to the corpus of knowledge that supports efficient constraint satisfaction and optimization procedures in real-world settings.

III. METHODOLOGY

The challenging issue of dispatch scheduling is to assign tasks to personnel or vehicles while reducing resource utilization and achieving objectives. MIP, or mixed-integer programming, is a useful technique for handling this problem. MIP assigns tasks to resources, uses binary variables to describe job start and finish times, and represents the dispatch scheduling problem as a mathematical optimization problem. The goal of the objective function is to shorten the total amount of time required to finish all jobs. The MIP solver looks at different combinations of task-resource allocations and start timings to determine the optimum solution. These approaches include cutting plane and branch and bound algorithms. In conclusion, MIP optimizes resource allocation and task completion by solving dispatch

scheduling as a mathematical optimization problem using decision variables, constraints, and algorithms.

Research goals, tasks, and resources must be defined, constraints must be taken into account, and the efficiency of MIP solvers must be evaluated when conducting a MIP for dispatch scheduling project. By selecting benchmark problems and adjusting parameter settings, an experiment may be built to assess the efficacy of several MIP solvers. It is crucial to use the right software tools, create software to develop benchmark problems and assess results, and create scripts to execute MIP solvers and gather performance information. The experiment is then carried out, data from each solver is collected, and the results are analyzed to gauge their efficacy. Understanding the MIP solver's performance can be accomplished by analyzing how it approaches the dispatch scheduling issue.

3.1 Reverse Engineering in Detail

In MIP, reverse engineering is looking at the solution process of a solver to comprehend its constraints, algorithms, and functions. It is critical to take confidentiality laws and intellectual property rights into account. It is crucial to secure the owner of the software's permission in order to protect a company's intellectual property rights. Open-source MIP solvers that are unrestricted by intellectual property laws can overcome ethical problems and make the study's results reproducible and verifiable by other researchers.

3.2 Data Gathering and Analysis

The efficiency of MIP solvers and dispatch scheduling issues can be evaluated through experiments and reverse engineering techniques using data analysis tools like Microsoft Excel. These resources offer summary statistics and aid in the organization of experimental data. Python is a powerful tool for data analysis, but it must take data security and privacy concerns into account. Before distributing the data, it is imperative to delete any sensitive or private information.

Mixed integer programming (MIP) is a potent method for job scheduling that optimizes work allocation, minimizes time, and lowers costs. MIP solvers can be used in many different industries because of their adaptability and versatility. In addition to defining the study question and creating the experimental design, researchers also need to run MIP solvers, monitor performance metrics, and conduct results analysis. Researchers can learn a lot about the performance of MIP solvers and boost task scheduling effectiveness by performing experiments, analyzing data, and taking ethical considerations into account.

Using cutting-edge methods like branch-and-bound, branch-and-cut, and heuristics, the experiment evaluates the performance of MIP solvers in actual optimization situations.

MIP uses a mathematical optimization library to implement above mentioned methods, these include Gurobi or open-source libraries such as SCIP. The main solving process of SCIP is depicted in Fig. 1.

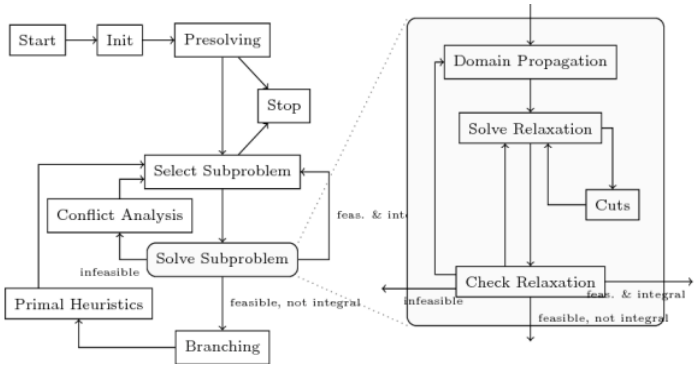

Fig. 1. Flow Graph of the main solving loop of SCIP

Another method of solving scheduling problems is by using Constraint Programming (CP) solvers. It is another effective method that can be utilized to solve dispatch scheduling issues. A summary of the underlying flow of a CP solver is depicted in Fig. 2.

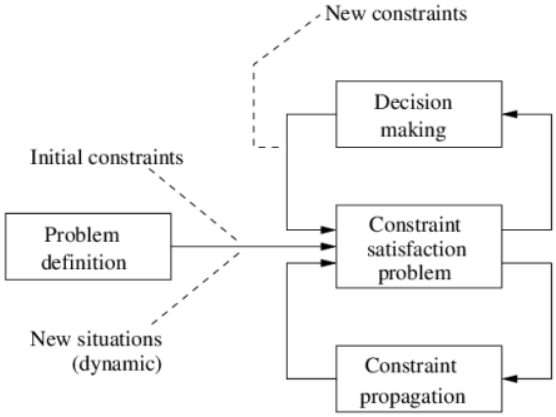

Fig. 2. Constraint programming flow

Constraint propagation refers to the process of reducing the domains of variables by applying constraints and their associated techniques [11]. Constraint propagation plays an integral role in narrowing down the search space and finding solutions efficiently.

The experiment analyzes system settings, constraints, variables, and algorithms to assess the performance of constraint solvers in Work task scheduling applications. With input variables and limitations created to closely resemble real-world events, the experiment can be carried out either manually or automatically. Data collection options are chosen throughout the design phase, and

pandas and Python Jupiter workspaces are used for analysis.

The design of the experiment variables, and definition of the outputs and outcomes will occur once the research objectives and limitations have been established. A detailed experiment plan, comprising a specific hypothesis, a list of required supplies and tools, a step-by-step technique, and a strategy for data collecting, will be created.

Data collection will be dependable, accurate, and valid since the experiment will be carried out in accordance with the concept and plan. The data will be analyzed, and findings and suggestions will be made on how to make Work task schedulers more effective and efficient when used with limitations.

3.3 Artifact

Agile method will be used to develop the Artifact or the system. Existing open-source software's or libraries with licenses are integrated to the artifact depending on its requirements. The research findings and suggestions will be constantly applied to improve the artifact. The artifact includes two separate APIs, a solver engine, and an interface for the end-users.

By utilizing input data, skill sets, and scheduling constraints, dispatching schedules are computer programs that optimize resource allocation and operational efficiency. The allocation of tasks or jobs to available resources, such as persons, vehicles, and equipment, is done using these restrictions. For scheduling purposes, the system considers job requests, location, and skill sets. In order to decrease downtime and raise service levels, the dispatching schedule generates a schedule or dispatch plan based on input data and constraints that can be changed in real-time.

The dispatch scheduler method uses a constraint solver to organize and structure input data, check constraints for conflicts, and prioritize constraints using heuristics or rules. The solver creates a list of alternative solutions and evaluates each one in light of the analysis and resolution of conflicts. Utilizing optimization strategies, the best answer is found. The answer is subsequently delivered, typically in the form of a list, report, or graphic representation. Resource allocation, scheduling, logistics, and optimization can all benefit from this powerful tool.

3.4 Experiment

The experiment intends to collect information on constraint solver performance in Dispatcher scheduling applications. A data sheet (ex: csv) is used to hold the gathered data, which can be collected manually or automatically. The data sheet is analyzed using Pandas data science tools and Python Jupiter workspaces. The study considers system settings, underlying algorithms, constraints, and variables. With careful regard for the system environment, the experiment can be run manually or automatically.
The steps of an experiment include setting research goals, designing the experiment, identifying obstacles, creating an experiment plan, carrying out the experiment, assessing the results, and coming to conclusions and making suggestions. These suggestions can be utilized to enhance the scheduler's functionality and utilization, thus maximizing its efficiency.

3.5 Reverse Engineering Approach

An effective way to learn about the underlying algorithms and performance traits of constraint solvers is by reverse engineering. To begin reverse engineering, one must first become familiar with the operation and intended application of the solver, study the documentation, and utilize the solver to become familiar with its capabilities. Next, locate crucial data structures and algorithms for resolving restrictions like data flow and source code. Benchmarking the solver on multiple input data sets will allow you to examine performance traits like time and memory needs. Try modifying and optimizing the solver to boost performance or fit it to new applications.

Reverse engineering may give rise to moral questions about things like privacy rights, intellectual property rules, and potential security holes in systems or products. It is essential to make sure that the reverse engineering technique respects privacy, avoids exploitation of weaknesses, and does not violate intellectual property rights. Furthermore, it is crucial to prevent the illicit duplication of products without permission. All things considered, reverse engineering is a difficult process that necessitates a profound comprehension of the solver's algorithms, data structures, and performance characteristics.

The research's technique is set up to systematically address the main research question, which is how system constraints affect constraint solver performance while scheduling optimization using various algorithms. The following steps and secondary goals make up the methodology:

Work task scheduling (WTS) and technician dispatching (TD) are the two scheduling domains that are being studied in detail in the first step, which is to identify the domain variables and constraints. To appropriately reflect the scheduling issues, domain variables that include both hard and soft constraints will be found. While soft constraints allow for flexibility in optimization, hard constraints set forth non-negotiable requirements. Documentation of pertinent restrictions, such as job

dependencies, resource availability, and time windows, will be done.

Using Mechanisms for Reverse Engineering to Find Underlying Algorithms By selecting Optaplanner and MiniCP as the constraint solvers, this stage tries to reveal the fundamental methods employed by each. The core algorithmic techniques used by these solvers will be determined using a mix of technical documentation analysis, published literature analysis, and, if accessible, source code analysis. Understanding how algorithmic elements affect solver behavior requires this realization.

Investigating the Use of Constraint Programming in Related Applications: To accomplish this goal, a thorough investigation of the use of constraint programming in related applications other than schedule optimization will be carried out. To comprehend how easily constraint programming techniques may be adapted and used in a variety of contexts, relevant academic material, case studies, and actual implementations will be examined. These scenarios' insights will help develop a more comprehensive understanding of constraint programming's use.

Analyzing the Performance of Selected Constraint Solvers: In this stage, empirical experiments are carried out to assess the effectiveness of the selected constraint solvers while taking into account various system constraints. Work task scheduling (WTS) and technician dispatching (TD) are the two identified scheduling domains where the experiment will be run, these domains are shown in fig. 3. Constraint programming (Optaplanner and MiniCP) and mixed integer programming (SCIP and CBC) solver categories will be evaluated.

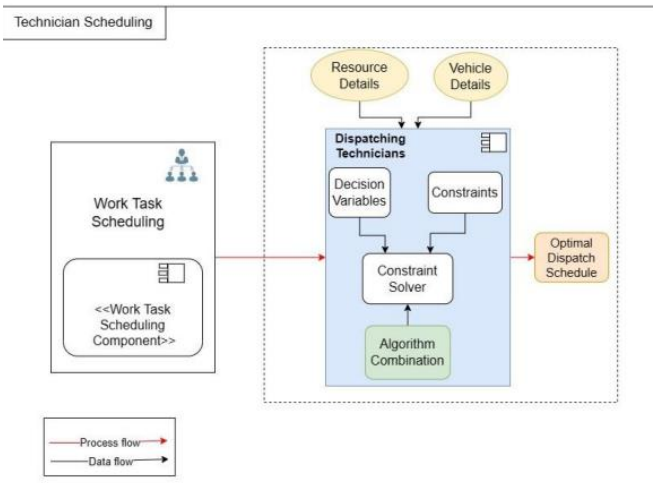

Fig. 3. Problem domains

3.6 Specific Case Methodology:

Work Task Scheduling (WTS): Use Optaplanner and MiniCP to provide constraint programming for WTS instances. Use SCIP and CBC to implement mixed integer programming for the same cases. System restrictions like computational capacity and task difficulty should be adjusted systematically. Analyze and contrast the runtime effectiveness, scalability, and solution quality of each solver under various constraint circumstances.

Utilize MiniCP and Optaplanner to apply constraint programming to instances of Technician Dispatching (TD). Implement mixed integer programming for identical instances using SCIP and CBC. Introduce various system constraints, such as geographic conditions and technician accessibility. Across various solvers and constraint changes, gauge and assess the solution quality, runtime effectiveness, and scalability. This methodology is used in the research in order to provide a thorough understanding of how different system constraints interact with constraint solvers while optimizing schedules with algorithmic options. The results of these tests will shed light on how solver performance, algorithm choice, and the limitations imposed by the problem and the computer environment are intertwined. The ultimate goal of this research is to provide practitioners with insightful information that will help them decide wisely in situations when schedule optimization is being used in the real world.

IV.RESULT AND DISCUSSION

Important conclusions were drawn from the examination of how system constraints affect the efficiency of constraint solvers during schedule optimization. The study used the solvers OptaPlanner and MiniCP for CP and SCIP and CBC for MIP, focusing on two well-known paradigms, Constraint Programming (CP) and Mixed Integer Programming (MIP).

4.1 Solver Performance Across System Constraints

Solvers for Constraint Programming (CP): OptaPlanner showed sensitivity to system constraints, with its performance changing as a result of the introduced constraints. OptaPlanner delivered reliable high-quality solutions with a variety of algorithm configurations when the system resources were plentiful. OptaPlanner's runtime did, however, significantly increase when constraints grew, particularly when CPU availability was constrained. Comparing global search algorithms to local search-based algorithms, the impact on solution quality was more pronounced for the former.

MiniCP behaved similarly to OptaPlanner, performing well under conditions with few constraints. MiniCP's runtime increased as system limitations grew more severe, albeit more slowly than with OptaPlanner. Across a range of algorithm configurations, the solver maintained a largely constant level of solution quality, with local search-based algorithms proving more resilient to limitations.

When subjected to system limitations, Mixed Integer Programming (MIP) Solvers such as SCIP showed noticeable performance variations. As constraints were added, its runtime considerably grew, especially when dealing with complex problem situations. Additionally, the quality of solutions declined, particularly as there were more variables and restrictions to consider. SCIP's performance was significantly impacted by limited computational resources, particularly memory. In contrast to SCIP, CBC behaved differently because it was more capable of adjusting to system limitations. CBC maintained more competitive performance across numerous scenarios, although still being impacted by restrictions. Because of its heuristics and branching techniques, it was better able to manage memory and CPU limitations, which reduced the impact on runtime and solution quality.

4.2 Algorithm Sensitivity to System Constraints

OptaPlanner and MiniCP both showed algorithm-specific sensitivity to system limitations, these solvers are known as constraint programming solvers (CP-solvers). In limited settings, local search algorithms typically outperformed global search algorithms with less noticeable runtime and solution quality erosion. This pattern indicates a better degree of adaptation to constrained computational resources in local search algorithms.

In Mixed Integer Programming (MIP) solvers, system restrictions had an impact on the performance of SCIP and CBC, with CBC showing higher resilience under some circumstances. Because of its heuristic-driven methodology and branching tactics, CBC was able to sustain competitive performance even with limited resources. The potential benefits of some MIP solvers in comparison with constrain solvers are highlighted by this research.

The findings highlight the complex interaction between solver performance, system restrictions, and algorithmic choices in schedule optimization. In both the CP and MIP paradigms, local search-based algorithms performed better when subjected to system restrictions. This is in line with the fundamental characteristics of local search algorithms, which emphasize gradual advancements and can adjust to changing resource availability. The adaptability of local search methods suggests that they are appropriate for use in practical settings where computational resources may be limited.

While global search algorithms performed worse when subjected to restrictions, CBC, a MIP solver, demonstrated the ability of MIP solvers to handle such situations successfully. By balancing exploration and exploitation, CBC was able to deliver competitive performance even when system resources were constrained. The results highlight the significance of choosing an algorithm depending on the features of the problem and the availability of resources. When attempting to balance solution quality and runtime efficiency, practitioners should take into account the unique algorithmic strategies and the available computational resources.

V.CONCLUTION

Investigation was how system limitations affected constraint solver performance during schedule optimization. Using the solvers OptaPlanner, MiniCP, SCIP, and CBC, two well-known paradigms, Constraint Programming (CP) and Mixed Integer Programming (MIP), were investigated. The study provided insightful information on the interactions between system constraints and algorithmic decisions that influence solver performance. The investigations showed that solver behavior changed depending on the severity of the system restrictions. Compared to global search algorithms, local search-based algorithms regularly demonstrated stronger ability to adapt to limited resources. As constraints became more severe, these local search techniques maintained more consistent solution quality and runtime efficiency. More study is needed in other paradigms such as Linear Programming, Evolutionary Algorithms, Gradient-based optimization etc. and their solvers.

IV. REFERENCES

[1] E Smith, J Frank, AK Jónsson, "Bridging the gap between planning and scheduling," *The Knowledge Engineering Review,* vol. 15, no. 01, pp. 47-83, 2000.

[2] D Hellmanns, L Haug, M Hildebrand, F Dürr, "How to optimize joint routing and scheduling models for TSN using integer linear programming," in *29th International Conference on Real-Time Networks and Systems*, 2021.

[3] Kotthoff, Lars, "Algorithm selection for combinatorial search problems: A survey," *Data mining and constraint programming: Foundations of a cross-disciplinary approach,* pp. 149-190, 2016.

[4] M Malawski, G Juve, E Deelman, J Nabrzyski, "Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," *Future Generation Computer Systems,* vol. 48, pp. 1-18, 2015.

[5] S Kadioglu, Y Malitsky, A Sabharwal, "Algorithm selection and scheduling," in *Principles and Practice of Constraint Programming–CP 2011: 17th International Conference*, Perugia, 2011.

[6] PE Bailey, A Marathe, DK Lowenthal, "Finding the limits of power-constrained application performance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015.

[7] Y Hamadi, E Monfroy, F Saubion, What is autonomous search?, Springer, 2011.

[8] K Smith-Miles, L Lopes, "Measuring instance difficulty for combinatorial optimization problems," *Computers & Operations Research,* vol. 39, pp. 875-889, 2012.

[9] HE Sakkout, M Wallace, "Probe backtrack search for minimal perturbation in dynamic scheduling," *Constraints,* vol. 04, pp. 359-388, 2000.

[10] O Alsac, J Bright, M Prais, B Stott, "Further developments in LP-based optimal power flow," *IEEE Transactions on Power Systems,* vol. 5, no. 3, pp. 697-711, 1990.

[11] F. Rossi, P. Van Beek, T. Walsh, Handbook of constraint programming, Elsevier, 2006.