Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

TOBIAS ACHTERBERG* J. CHRISTOPHER BECK** (EDS.)

CPAIOR 2011
Late Breaking Abstracts

* IBM, Software Group, CPLEX Optimization
** Department of Mechanical & Industrial Engineering, University of Toronto, Toronto, Canada

ZIB-Report 11-20 (May 2011)

Preface

The Eighth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2011) was held in Berlin, Germany, May 23-27, 2011.

The conference is intended primarily as a forum to focus on the integration and hybridization of the approaches of Constraint Programming (CP), Artificial Intelligence (AI), and Operations Research (OR) technologies for solving large scale and complex real life combinatorial optimization problems. CPAIOR is focused on both theoretical and practical, application-oriented contributions.

As part of the conference a call was made for late breaking abstracts for presentation at the conference. These abstracts were meant to represent work in process, recent work, or work appearing in other academic areas but of interest to the CPAIOR community. A total of 19 submissions were received of which 16 were selected by the program chairs for presentation. This document is a compilation of the presented abstracts.

The submissions were not peer-reviewed but rather selected on the basis of interesting ideas. The authors of the individual abstracts retain all rights and copyrights.

The formal proceedings of CPAIOR 2011, which do not include the abstracts in this technical report, can be found in the following publication:

> Achterberg, T. & Beck, J.C., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: Proceedings of the 8th International Conference*, Lecture Notes in Computer Science 6697, Springer, 2011.

Thanks to the Department of Scientific Information of the Zuse Institute Berlin, video recordings of the presentations of CPAIOR 2011 were made. They can be found on the CPAIOR 2011 webpage at `http://cpaior2011.zib.de`. We would like to specially thank Wolfgang Dalitz and the Web Technology and Multimedia group for creating this valuable record of the conference.

The staff at Zuse Institute Berlin did an outstanding job providing administrative support, making sure the money was in the right place at the right time, and in handling the registrations. In particular, we would like to thank Annerose Steinke, Sylke Arencibia, Sybille Mattrisch, Bettina Kasse.

A special thanks goes to the conference chairs, Timo Berthold, Ambros Gleixner, Stefan Heinz, and Thorsten Koch for the organization and substantial efforts on sponsorship, publicity, logistics, and all the other things that have to happen behind the scenes to make a conference work.

Finally, we would like to thank the sponsors who make it possible to organize this conference:

DFG Research Center Matheon, Zuse Institute Berlin, the Association for Constraint Programming, SAS, IBM, AIMMS, Gurobi Optimization, FICO, the Institute for Computational Sustainability, GAMS, IVU Traffic Technologies AG, MOSEK Optimization, National ICT Australia, Jeppesen, the ABB Group, atesio GmbH, and ProCom GmbH.

May 2011

Tobias Achterberg
J. Christopher Beck

Table of Contents

Satisfiability Test for the `energy` Constraint

Christian Artigues, Pierre Lopez, and William Mangoua Sofack

CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse Cedex 4, France
Université de Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France

We define a general `energy` constraint[1] abstracted from an industrial application considering electricity modulation for scheduling of melting operation [1]. We consider a set of n tasks $A = \{1, 2, \ldots, n\}$ and a resource of constant capacity B. Let $\mathcal{T}$ denote a continuous or discrete scheduling horizon. Each task has a required energy W_i, a minimum resource requirement b_i^{min} and a maximum resource requirement b_i^{max}, a release date r_i, and a due date d_i. The global `energy` constraint holds iff one can find for each task $i \in A$, a start time st_i, a finish time ft_i and a resource usage profile $b_i(t)$, $\forall t \in \mathcal{T}$ verifying:

$$st_i \geq r_i \wedge ft_i \leq d_i \quad \forall i \in A, \tag{1}$$

$$b_i^{min} \leq b_i(t) \leq b_i^{max} \quad \forall i \in A, t \in [st_i, ft_i[, \tag{2}$$

$$\sum_{i \in A} b_i(t) \leq B \quad \forall i \in A, t \in \mathcal{T}, \tag{3}$$

$$\int_{t=st_i}^{ft_i} b_i(t) = W_i. \tag{4}$$

The `energy` constraint states that each task has to be scheduled inside its time window (1) and, once started, must respect at each time the boundaries on its resource usage (2). Moreover, the integral of its resource profile must be equal to the required energy (4), while, at each time, the cumulative resource usage of the tasks must respect the resource capacity (3). We define the task duration as $p_i = ft_i - st_i$. In the discrete time case, assuming unit time periods, (4) can be written $\sum_{t=st_i}^{ft_i - 1} b_i(t) = W_i$. If $\mathcal{T}$ is discrete, $b_i^{min} = b_i^{max} = b_i$ and $W_i = p_i b_i$, we have the particular case of the `cumulative` constraint [3]. The consequence is that deciding whether the `energy` constraint is satisfiable is NP-complete. If $\mathcal{T}$ is discrete and $b_i^{min} = 0$, $b_i^{max} = B$ we obtain the (polynomially solvable) fully elastic constraint [2]. If starting and finishing times are fixed and for a discrete time horizon, we have the following results.

Theorem 1. *For fixed $(st_i, ft_i)_{i \in A}$ and discrete $\mathcal{T}$, satisfiability of the* `energy` *constraint can be checked polynomially in function of $|\mathcal{T}|$ and n.*

Proof (sketch). If $b_i(t)$ takes continuous values, (2-4) become a linear program. Otherwise, the problem can be solved as a maximum flow problem. □

[1] This paper is an abstract version of the short paper that was submitted to CPAIOR 2011 (and rejected). Theorems 1 and 2 were added as a partial answer to a question of a referee.

Theorem 2. *For fixed $(st_i, ft_i)_{i\in A}$, discrete $\mathcal{T}$ and continuous $b_i(t)$, satisfiability of the* `energy` *constraint can be checked polynomially in function of n.*

Proof (sketch). Consider the increasing series $(t_q)_{q=1,\ldots,Q}$ of distinct start and end time values ($Q \leq n^2$) and a feasible solution $b_i(t)$. For each t, we have $\sum_{i\in A} b_i(t) \leq B$. Summing over all $t \in \{t_q, \ldots, t_{q+1} - 1\}$ yields

$$\sum_{t=t_q}^{t_{q+1}-1} \sum_{i\in A} b_i(t) \leq (t_{q+1} - t_q)B$$

and consequently,

$$\sum_{i\in A} \sum_{t=t_q}^{t_{q+1}-1} b_i(t)/(t_{q+1} - t_q) \leq B.$$

Hence for each $t \in \{t_q, \ldots, t_{q+1}-1\}$, b_{it} can be replaced by $b'_{it} = \sum_{t=t_q}^{t_{q+1}-1} b_i(t)/(t_{q+1}-t_q)$ while keeping the solution feasible. It follows there is no need to change the amount of resource allocated to a task at time point that does not coincide with the start or the end of another task. It follows that by replacing $b_i(t)$ by decision variable $b_{iq}/(t_{q+1} - t_q)$ for $t \in \{t_q, \ldots, t_{q+1} - 1\}$, in the linear program (2-4), where b_{iq} is the amount of energy allocated to i in interval q, yields a linear program with a number of variables and constraints polynomial in n. □

When starting and finishing times must be determined, as the `energy` constraint generalizes the `cumulative` constraint, deciding whether an `energy` constraint has a solution is NP-complete. It is consequently of interest of establishing necessary feasibility conditions, i.e. an incomplete satisfiability such we are sure the constraint cannot be satisfied is the test returns `false` while we can not conclude on the constraint satisfiability the test returns `true`. For the `cumulative` constraint, the energetic reasoning is a successful incomplete satisfiability test used in commercial solvers [2,3]. It can be stated as follows. Consider the minimum energy consumption of a task i over an interval $[t_1, t_2]$, ignoring other tasks. This value is denoted by $\underline{w}(i, t_1, t_2)$. Clearly, the `energy` constraint cannot be satisfied if

$$\exists t_1, t_2 \in \mathcal{T}^2, t_2 > t_1, \sum_{i\in A} \underline{w}(i, t_1, t_2) > B(t_2 - t_1) \tag{5}$$

In [2], it was shown that test (5) can be restricted for the `cumulative` constraint to a set of dominant intervals of polynomial cardinality. We generalize below these results to the `energy` constraint.

Theorem 3. *For the* `energy` *constraint, test (5) can be performed in strongly polynomial time.*

Proof (sketch). First, we show that for all $i \in A$, $\underline{w}(i, t_1, t_2)$ is a 2D continuous piecewise linear function: $\exists q \in \{0, 1, 2, 3, 4, 5\}$, $\underline{w}(i, t_1, t_2) = f_q(i, t_1, t_2)$ where linear functions $f_q(i, t_1, t_2)$ are defined below.

q	$f_q(i,t_1,t_2)$	q	$f_q(i,t_1,t_2)$
0	0	3	$W_i - b_i^{\max}(t_1 - r_i)$
1	W_i	4	$W_i - b_i^{\max}(t_1 - r_i) - b_i^{\max}(d_i - t_2)$
2	$W_i - b_i^{\max}(d_i - t_2)$	5	$b_i^{\min}(t_2 - t_1)$

We now define the slack of an interval $SL(t_1,t_2) = B(t_2-t_1) - \sum_{i \in A} \underline{w}(i,t_1,t_2)$. Equivalently to (5), the energy constraint is not satisfiable if $\exists [t_1,t_2]$ such that $SL(t_1,t_2) < 0$. $SL(t_1,t_2)$ is also a 2D continuous piecewise linear function and its inflexion lines are the union of the inflexion lines of individual task functions $\underline{w}(i,t_1,t_2)$. For each task i, there is a constant number of inflexion lines. Hence there are $O(n)$ inflexion lines for $SL(t_1,t_2)$. Inside the polytopes defined by the inflexion lines, the slack is linear. The minimum of $SL(t_1,t_2)$ is reached on an extreme point of one of these polytopes, i.e. an intersection of two inflexion lines. Hence there are $O(n^2)$ extreme points to which test (5) can be restricted. □

From Theorem 3, an $O(n^3)$ algorithm can be derived to perform test (5) for all tasks. When $b_i^{min} = b_i^{max} = b_i$, the dominant intervals for the cumulative constraint [2] are obtained. An open question is whether the test is also a sufficient feasibility condition when start and end time are fixed, i.e. if it subsumes the linear program. These preliminary results are interestingly general as Theorem 3 makes no hypothesis for functions $b_i(t)$. More precisely, the resource demands and the time horizon can be continuous or discrete. Furthermore, for the discrete time case, Theorems 1 and 2 provide in addition a way of restricting the search to feasible start and end times since the $b_i(t)$ can be obtained (pseudo-)polynomially (and even polynomially for continuous $b_i(t)$) for fixed st_i and ft_i. For continuous $b_i(t)$, Theorem 2 provide also a dominance rule for resource amount change times. As a short-term follow-up of this work, we are working in two research directions. The first one consists in designing an intelligent method for enumerating the extreme points of the slack polytopes to obtain an efficient energy reasoning algorithm. For the second direction, a complete tree-search method embedding energy reasoning, linear programming and/or network flow algorithms will be proposed to obtain a feasible solution or to prove that no solution exists.

References

1. C. Artigues, P. Lopez, and A. Hait: The energy scheduling problem: Industrial case study and constraint propagation techniques. International Journal of Production Economics, doi:10.1016/j.ijpe.2010.09.030.
2. P. Baptiste, C. Le Pape, and W. Nuijten: Satisfiability tests and time-bound adjustments for cumulative scheduling problems. Annals of Operations research, 92, 305–333 (1999)
3. P. Lopez and P. Esquirol: Consistency enforcing in scheduling: A general formulation based on energetic reasoning. In: 5th International Workshop on Project Management and Scheduling, pp. 155–158, Poznan, Poland (1996)

Learning Graphical Models for Algorithm Configuration

Mauro Birattari[1], Marco Chiarandini[2],
Marco Saerens[3], and Thomas Stützle[1]

[1] IRIDIA, Université Libre de Bruxelles, mbiro|stuetzle@ulb.ac.be
[2] IMADA, University of Southern Denmark, marco@imada.sdu.dk
[3] Machine Learning Group, Université catholique de Louvain,
marco.saerens@uclouvain.be

Algorithms for solving optimization problems have typically a large number of inherent parameters, often related to heuristic choices, that cannot be assessed by theoretical analysis only. Examples are branching rules in a branch and bound algorithm, the type of neighborhood in a local search heuristic, the rate of evaporation of pheromone in ant colony optimization or algorithmic modules and their order in hybrid solvers. Moreover, the choice of parameter settings depends on the input data and may vary from one type of input data to another. The configuration of these parameters is carried out by executing computational experiments on simulated or real-life data. This procedure can be burdensome due to the large number of possible candidate configurations to test and the high cost in time the process may require.

In the recent years, a considerable amount of research within computer science has been concerned with the development of appropriate methods for the automatic configuration of optimization algorithms. Given a problem, a set of solution algorithms with relevant parameters exposed and a description of the input data, the goal is finding the setting of parameter values that is most likely to perform best.

The main stream of this research is an interdisciplinary approach with the field of *statistics*. The aspects of statistics that are appealing in this context are the objectivity provided by the mathematical framework and the existence of methods for estimating the contribution of factors while minimizing the number of experiments to execute. In this latter category we find: advanced techniques for experimental designs, response surface modelling and sequential testing.

In this work, we study a different approach to the task of automatic configuration based on probabilistic graphical models. Graphical models are well known abstractions for representing knowledge developed in the field of machine learning. This representation in computer readable form is manipulated by various algorithms to learn and make inference. Graphical models are well suited to handle uncertainty due to different causes, such as lack of knowledge or inherent stochasticity of events. Moreover, they can cope with situations where all possibilities cannot be considered because they are simply too many. This is achieved by shifting the analysis of the different possibilities into the framework of probability calculus and focusing on their likelihood.

In algorithm configuration, the nodes of the graphical model represent the parameters to tune and connections the dependencies between these parameters. Each node is a random variable with a probability distribution that reflects the knowledge on the parameter that it represents. We learn the configuration of parameter values that is most likely to perform the best, in the same way as expert systems learn the most probable explanation to the input evidence. In our specific context, this requires a further step: recognizing a stochastic optimization problem and solving it by rare event simulation. In other terms, we modify the joint probability distribution of the model in order to make more likely to infer good configurations of parameter values. We achieve this by sampling configurations from the network and learning on those that perform above an adaptive threshold. Bayesian calculus is used to learn the joint distribution revising previous knowledge on the basis of new evidence.

The method proposed can treat all types of parameters arising in algorithm design: categorical, discrete and continuous. Moreover, it can deal with probabilistic dependencies as well as deterministic dependencies, that is, nesting of parameters under choices of parent parameters.

We tested the method on heuristics for the traveling salesman problem, including combinations of construction heuristic, local search and ant colony optimization. Preliminary results seem to indicate that the performance of the proposed method is competitive against iterated F-race, a previously proposed configuration procedure that received wide interest in the literature. In addition, it provides information on which parameters affect most the results and on their interaction.

Future work includes testing the configuration procedure on exact methods such as constraint programming and integer programming. Some features of the method make it appealing in these contexts. It is possible to embed a priori knowledge provided by experts and/or learned from initial tests on input data of small size. The knowledge inferred is then tested and corrected on larger instances that are computationally more expensive. Further, a parallel implementation of the configuration procedure makes it possible to truncate runs of configurations as soon as a desired number of configurations on which we want learning to occur have found a solution.

Comparing Integer Programming and Constraint Programming for a Flow Shop Lot Streaming Problem

Rahime Sancar Edis[1], Emrah B. Edis[2], and Ceyda Oguz[3]

[1] Celal Bayar University-Department of Industrial Engineering
Muradiye, 45140, Manisa, Turkey
rahime.edis@bayar.edu.tr
[2] Dokuz Eylül University – Department of Industrial Engineering
Buca, 35160, Izmir, Turkey
emrah.edis@deu.edu.tr
[3] Koç University – Department of Industrial Engineering
Sariyer, 34450, Istanbul, Turkey
coguz@ku.edu.tr

Lot streaming (LS) is a technique that splits the production lot into sublots, and schedules these sublots in an overlapping way on the machines in order to accelerate the process of orders and to improve the overall system performance. In this study, an LS problem in a multi product, multi machine flow shop is considered with equal sublot type, i.e., all the sublots of a product are of the same size. The objective is to minimize the makespan, i.e., completion time of all sublots on all machines.

Since the sublot sizes are known a priori in equal sublot case, the considered multi product LS problem requires only sequencing the products through the machines. In this study, we first give a mixed integer programming (MIP) model of the entire problem. However, this MIP model could not give efficient results especially for large problem sizes. On the other hand, it is well known that constraint programming (CP) techniques may provide efficient results for sequencing and scheduling problems [1]. Therefore, for the considered problem, a CP model is developed in IBM OPL IDE 6.3TM [2] using its special structures designed for scheduling problems, e.g., interval variables, noOverlap constraints etc. In the proposed CP model, we use interval variables to identify the sublots and products processed on each machine. In order to obtain efficient results, a number of different search phases are also identified and evaluated in the CP model. The search phases constructed based on the interval variables have provided better performance than the ones based on sequencing variables.

For the computational study, we classified the test instances into three groups: small, medium and large problems. The number of products was taken as 5 and 7, 10 and 15, 20 and 30, for small, medium and large sized problems, respectively. For all experimental points, both the number of machines and the number of sublots were taken as 5 and 10. Since the MIP model could not produce even feasible results for medium and large sized problems; to evaluate the performance of the proposed CP model, we adapted a well known heuristic algorithm, i.e., Nawaz, Encore and Ham (NEH) [3], to the LS problem on hand [4]. All three solution

approaches were run on a Core 2 Duo 2.2 GHz, 2 GB RAM computer. For each experimental point, we solved 10 test instances of small sized test problems, and 5 test instances for the medium and large sized problems. A run-time limit of 1000 seconds was set for all solution approaches. Figure 1 illustrates the performance of the solution approaches for each experimental point in terms of the average deviation from the corresponding best makespan values.

In Figure 1, the experimental points are given in the x-axis. For example, 15-5-10 represents the experimental point with 15 products, five machines and 10 sublots.

For the small sized test problems for which the MIP model is able to produce optimal results, the proposed CP model performs as good as the MIP model in terms of the solution quality, while the adapted NEH algorithm results in considerable deviations from the optimal values.

On the other hand, for the medium and large problem sizes where the MIP model could not handle the test instances, the computational results generally indicate that the proposed CP model produces better results than the adapted NEH algorithm. For medium sized problems, CP model outperforms the NEH algorithm in all experimental points. In terms of large sized problems, CP model also provides better performance for almost all experimental points. For only 30-product and 10-machine test problems, the performance of the NEH algorithm is slightly better than the proposed CP model. This may be due to a significant increase in the number of variables as well as their domain ranges.

Consequently, for the considered LS problem, the proposed CP model generally provides efficient solutions.

For the ongoing research, we are studying to improve the performance of the proposed CP model and to extend this study to the consistent sublot types in which the sublots of a product may take different sizes on the same machine. Since this extension incorporates sublot allocation decisions along with the sequencing ones, decomposition based solution techniques may provide efficient results.

Keywords: lot streaming, flow shop, integer programming, constraint programming.

References

1. J.J. Ahire, S.L. Kanet, and M.F. Gorman: Constraint programming for scheduling. *In: J.Y-T. Leung (Ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis (Chapter 47).* USA: CRC Press (2004).
2. IBM ILOG OPL V6.3, IBM ILOG OPL Language User's Manual, IBM Corporation, (2009).
3. M. Nawaz, E.E. Encore, and I. Ham: A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *OMEGA*, 11, 91-95 (1983).
4. R.S. Edis: Tabu search based solution approaches for lot streaming problems in flow shops. *PhD. Thesis*, Dokuz Eylül University, Graduate School of Natural and Applied Sciences, Izmir, Turkey (2009).

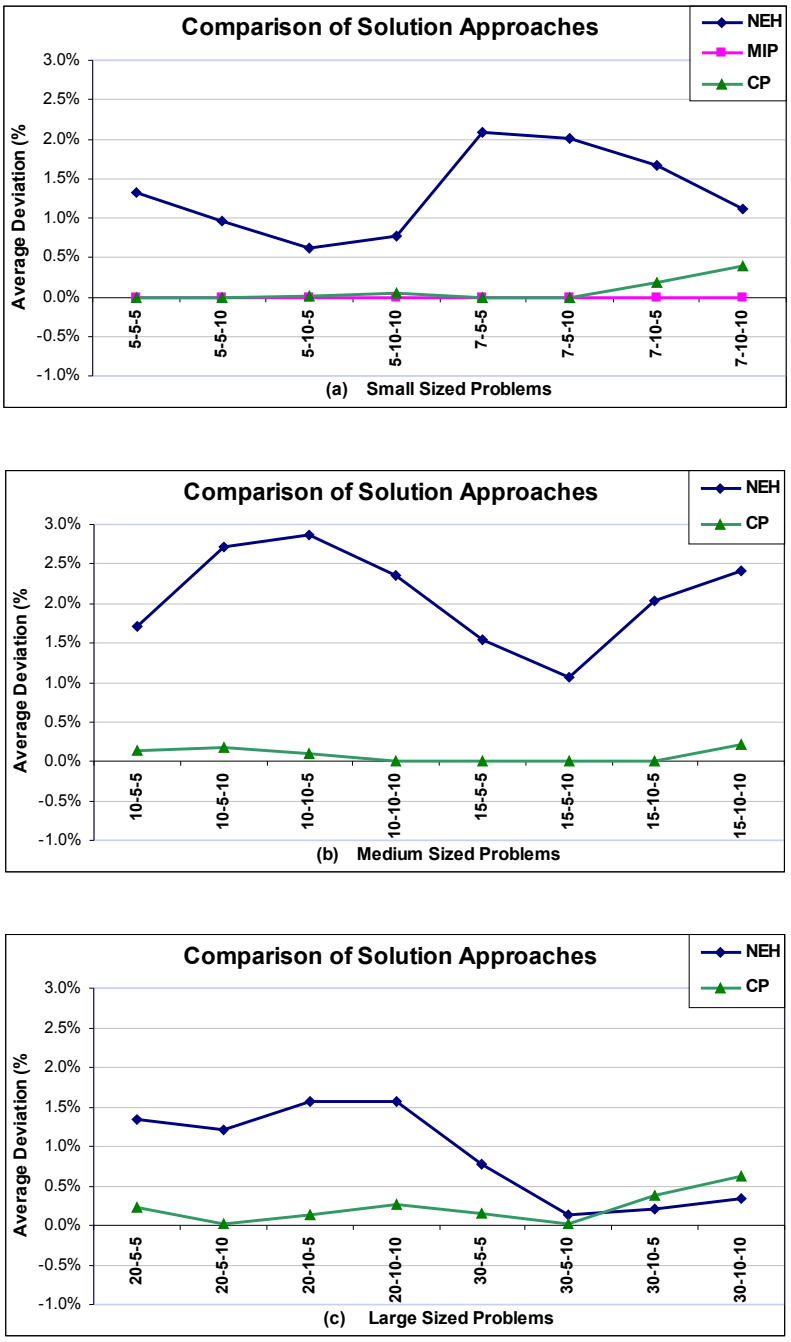

Fig. 1. Comparison of the solution approaches

Three ideas for the Quadratic Assignment Problem

Matteo Fischetti, Michele Monaci, and Domenico Salvagnin

DEI, University of Padova, via Gradenigo 6/A, 35131 Padova, Italy
{matteo.fischetti,michele.monaci,domenico.salvagnin}@unipd.it

1 The challenge

The NP-hard (and notoriously very difficult in practice) *Quadratic Assignment Problem* (QAP), in its Koopmans and Beckmann [9] form, can be sketched as follows; see, e.g., Burkard, Dell'Amico and Martello [4] for details.

We are given a complete directed graph $G = (V, A)$ with n nodes and n^2 arcs along with a set of n facilities to be assigned to its nodes. In what follows, indices $i, j \in V$ always correspond to nodes, indices $u, v = 1, \cdots, n$ to facilities, $b_{ij} \geq 0$ is a given *distance* from node i to node j, $a_{uv} \geq 0$ is a given required *flow* from facility u to facility v, and c_{iu} is a given fixed cost for assigning facility u to node i. By using binary variables $x_{iu} = 1$ iff facility u is assigned to node i, QAP can be stated as the following quadratic 0-1 problem:

$$\min \sum_i \sum_u \sum_j \sum_v a_{uv}\, b_{ij}\, x_{iu}\, x_{jv} + \sum_i \sum_u c_{iu} x_{iu} \tag{1}$$

$$\sum_i x_{iu} = 1 \ \forall u$$

$$\sum_u x_{iu} = 1 \ \forall i$$

$$x_{iu} \geq 0 \ \text{ and } \ \text{integer} \quad \forall i, u \tag{2}$$

In spite of its simple definition, QAP is among the most difficult optimization problems arising in practice, and its study attracted a large amount of research. A QAP feature that challenged us is that a collection of (apparently very small) test-cases is available in the QAPLIB [5], that cannot be solved by the current state-of-the-art algorithms even by allowing for tremendous computing power. E.g., an instance with just $n = 30$ such as `tho30` was solved only recently and required the equivalent of 8,997 days of computation on a distributed grid [3]—and many similar instances are still unsolved nowadays.

Among the difficult cases, we concentrated on the so-called `esc` instances introduced in [7]. As reported in QAPLIB, instances `esc32a`, `esc32b`, `esc32c`, `esc32d`, `esc32h`, `esc64a`, and `esc128` are still unsolved by using any published method. It is fair however to mention that the "What's new" QAPLIB section at `http://www.seas.upenn.edu/QAPLIB/` states that in January 2011 Axel Nyberg and Tapio Westerlund at Abo Akademi University in Finland announced the solution of `esc32a`, `esc32c`, `esc32d`, and `esc64a` (plus `tai64c`). Instance `esc32c` took 9 hours on a single PC using gurobi 3.0 (default settings), whereas `esc32d` took about 35 hours. As far as we know, however, no paper describing this

method is available nor circulated in any way, so no comparison with our own approach can be made.

We next mention very briefly the three main steps that qualified as a breakthrough for our approach, namely (1) exploiting symmetries, (2) using a manageable MILP formulation, and (3) designing an ad hoc branch-and-cut solution algorithm.

So far our method was able to solve, in a matter of seconds or minutes on a single PC, all the easy cases (all `esc16*` plus `esc32e` and `esc32g`). The three previously-unsolved `esc32c`, `esc32d` and `esc64a` were solved in roughly half an hour, all together, on a single quadcore PC. We also report the solution of the previously-unsolved `tai64c`, again within reasonable computing time. We are currently attacking the remaining cases, including "the big fish" `esc128` for which we found an improved lower bound of 56 out of 64 (the previous best lower bound was 2).

LATEST NEWS. We finally succeeded in solving `esc128` to proven optimality—to our great surprise, by exploiting a facility-decomposition argument this task took just a few seconds of our quadcore PC. According to [5], `esc128` is the largest QAP instance ever solved by an exact method.

Step One: Exploiting symmetry

In his survey [2], Anstreicher observed that "careful consideration of the structure present in the larger `esc` instances is likely to be important in attempts to solve these problems to optimality". Indeed, `esc` instances are known to be highly symmetrical, and important attempts to deal with this property have been made in the literature, including those in [8] and [6]. Of course, one could cope with symmetry by imposing additional conditions in the model, or by using a modified branching strategy such as isomorphism pruning [10] or orbital branching [11]. But can one actually *take advantage* of symmetry?

We will give a positive answer to the above question, and we will describe a new procedure to remove a main source of symmetry by actually *reducing* the instance size. Our technique can be viewed a generalization of a method proposed by Kaibel [8] for dealing with the case where $k \leq n$ facilities have to be assigned to n nodes, and leads to a dramatic performance improvement on certain types of instances such as the `esc` ones.

Step Two: Working with a handy MILP model

Most MILP models for QAP work with additional 0-1 variables $y_{iujv} = x_{iu}x_{jv}$ that are used to linearize the quadratic objective function—the Adams-Johnson model [1] being perhaps the best-known such formulation. These kinds of models require about n^4 variables and n^3 constraints, so they become huge even for medium-size instances.

As our ultimate goal is to solve `esc128`, we decided to address more scalable models that do not become hopeless for large instances. We exploited a

MILP formulation with just $O(n^2)$ variables and constraints that gives a good compromize between bound quality and solution speed of its LP relaxation.

Our first quick shot was just to generate our MILP model to solve it through a black-box commercial MILP solver—IBM Cplex 12.2 in our case. We provided a branching-priority input file where the branching priority of variable x_{iu} is defined as a measure of the impact of x_{iu} into our model, as recently suggest by Chinneck and co-authors [12,13]. This feature turned out to be instrumental for the success of our method.

Table 1 reports results obtained by using Cplex 12.2 for solving to proven optimality some easy and hard `esc` instances (plus `tai64c`). The experiments were performed on a single quadcore PC Intel i7-860 running at 2.8 GHz and equipped with 8GB ram. According to preliminary tests, Cplex has no difficulty in finding the optimal solution (within negligible computing time), while cuts seem to be counterproductive. Hence both cuts and heuristics were disabled in our final runs, and no upper cutoff was set. Instead, we found highly beneficial to use the most aggressive setting for symmetric reductions (level 5). In all cases, the optimal solutions turned out to be of the same value as those reported in QAPLIB.

Table 1. Optimality proved by IBM Cplex 12.2 in interactive mode (8 threads on a quadcore Intel i7-860 PC@2.8GHz with 8GB RAM). Instances marked by $*$ are reported as unsolved in QAPLIB.

Instance	n	OPT	CPU sec.s	#nodes
`esc16a`	16	68	0.48	5,427
`esc16b`	16	292	4.54	71,109
`esc16c`	16	160	170.81	2,633,093
`esc16d`	16	16	0.55	8,793
`esc16e`	16	28	0.06	412
`esc16f`	16	0	0.00	0
`esc16g`	16	26	0.09	484
`esc16h`	16	996	0.23	2,849
`esc16i`	16	14	0.19	3,122
`esc16j`	16	8	0.05	112
`esc32c`*	32	642	9,383.88	68,300,086
`esc32d`*	32	200	6,264.67	21,316,779
`esc32e`	32	2	0.05	64
`esc32g`	32	6	0.05	585
`esc64a`*	64	116	613.40	1,644,709
`tai64c`*	64	1,855,928	20,512.23	1,216,074,081

Step Three: Designing a branch-and-cut algorithm

Our next (still ongoing) attempt has been the design of an ad-hoc Branch&Cut code where additional cuts are generated, on the fly, to improve lower bound quality at some branching nodes. In addition, a specialized orbital branching [11] scheme has been implemented. Table 2 reports results when using a preliminary version of our code. Note that for `esc128` we were able to prove a lower bound of 56 (out of 64 or less), which is much tighter than the previous bound of 2 reported in the literature.

Table 2. Optimality proved by our specialized Branch&Cut code built on top of IBM Cplex 12.2 (8 threads on a quadcore Intel Xeon PC@3.2GHz with 16GB RAM). All these instances are reported as unsolved in QAPLIB.

Instance	n	OPT	CPU sec.s	#nodes
`esc32c`	32	642	1375.3	4,094,331
`esc32d`	32	200	495.9	719,887
`esc64a`	64	116	82.5	142,945

Table 3. Lower and upper bounds for still unsolved `esc` instances. LB and UB are from QAPLIB, ourLB is the current lower bound available after about 3 wall-clock days of enumeration through our Branch&Cut code (16 threads on a IBM Power7 CPU @3.07GHz with 128GB RAM).

Instance	n	LB	ourLB	UB
`esc32a`	32	103	108	130
`esc32b`	32	132	134	168
`esc32h`	32	424	380	438
`esc128`	128	2	56	64

Acknowledgments

Ongoing research supported by the *Progetto di Ateneo* on "Computational Integer Programming" of the University of Padova. We thank Gianfranco Bilardi and

Enoch Peserico for the use of the computers of the Center of Excellence "Scientific and Engineering Applications of Advanced Computing Paradigms".

References

1. W.P. Adams and T.A. Johnson. Improved linear programming-based lower bounds for the quadratic assignment problem. In *Proceedings of the DIMACS Workshop on Quadratic Assignment Problems*, volume 16 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 43–75. American Mathematical Society, 1994.
2. K. M. Anstreicher. Recent advances in the solution of quadratic assignment problems. *Mathematical Programming*, 97(1–2):27–42, 2003.
3. K.M. Anstreicher, N.W. Brixius, J.-P. Goux, and J. Linderoth. Solving large quadratic assignment problems on computational grids. *Math. Program.*, 91(3, Ser. A):563–588, 2002.
4. R.E. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM, 2009.
5. R.E. Burkard, S. Karisch, and F. Rendl. QAPLIB – A quadratic assignment problem library. *ejor*, 55:115–119, 1991. www.opt.math.tu-graz.ac.at/qaplib/.
6. E. de Klerk and R. Sotirov. Exploiting group symmetry in semidefinite programming relaxations of the quadratic assignment problem. *Mathematical Programming*, 122(2):225–246, 2010.
7. B. Eschermann and H. J. Wunderlich. Optimized synthesis of self-testable finite state machines. In *20th International Symposium on Fault-Tolerant Computing (FFTCS 20)*, 1990.
8. V. Kaibel. Polyhedral combinatorics of quadratic assignment problems with less objects than locations. In Robert Bixby, E. Boyd, and Roger Ros-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 409–422. Springer Berlin / Heidelberg, 1998.
9. T.C. Koopmans and M.J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.
10. F. Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94(1):71–90, 2002.
11. J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio. Orbital branching. *Mathematical Programming*, 126(1):147–178, 2011.
12. J. Patel and J.W. Chinneck. Active-constraint variable ordering for faster feasibility of mixed integer linear programs. *Mathematical Programming*, (110(3)):445–474, 2007.
13. J. Pryor and J.W. Chinneck. Faster integer-feasibility in mixed-integer linear programs by branching to force change. *Computers & OR*, 38(8):1143–1152, 2011.

Explanation Algorithms for Cumulative Scheduling*

Stefan Heinz[1] and Jens Schulz[2]

[1] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
heinz@zib.de
[2] Technische Universität Berlin, Institut für Mathematik, Straße des 17. Juni 136, 10623 Berlin, Germany
jschulz@math.tu-berlin.de

1 Introduction

In cumulative scheduling, conflict analysis is one of the key ingredients to solve these problems efficiently, see [2,5,7]. Thereby, the computational complexity of explanation algorithms that 'explain' infeasibilities or bound changes plays an important role. Their role is even more substantial when we are faced with a backtracking system where explanations need to be constructed on the fly.

In this talk we present complexity results for computing minimum-size explanations for the propagation algorithms time-tabling, edge-finding, and energetic reasoning. Due to the hardness results, we present optimal and heuristic approaches to deliver explanations. Our computational results show that minimum-size explanations drastically decrease the number of nodes in a branch-and-bound tree search and reduce the computation times by one-half.

2 Problem Description

In cumulative scheduling we are given a set of jobs that require a certain amount of resources. In our case, the resources are renewable with a constant capacity and each job is non-interruptible with a fixed processing time and demand request for one or several resources. A resource can be, for example, a group of workers with the same specialization, a set of machines, or entities like power supply. Additionally, each job has an earliest start time and a latest completion time. The goal is to find a feasible start time for each job such that the available capacities of the resources are respected at any point in time.

Cumulative scheduling problems have been tackled with techniques from constraint programming (CP), integer programming, or satisfiability testing (SAT). Hybrid approaches have been developed which combine methods from these areas. Currently, the best results are reported by a hybrid solver which uses CP and SAT techniques [7]. However, there are still instances with 60 jobs and four cumulative constraints published in the PSPLib [6] that resist to be solved to proven optimality.

* Supported by the DFG Research Center Matheon *Mathematics for key technologies* in Berlin.

3 Conflict Analysis

Various exact approaches use a branch-and-bound approach to solve these $\mathcal{NP}$-hard problems. Among them are SAT solvers that make explicit use of infeasible subproblems. They perform non-chronological backtracking or derive *no-goods* to speed up the search. As can be seen in recent publications, this *conflict analysis* plays an important role to solve cumulative scheduling problems efficiently [2,5,7].

Conflict analysis takes place under the following circumstances. During branch-and-bound search the lower and upper bounds of variables are updated by various propagation algorithms or simply by branching decisions. A subproblem becomes infeasible, e.g., because the lower bound of the start time variable of some job can be updated to a value larger than the current upper bound, because the resource demand of all jobs for some interval is larger than the available capacity, or because a relaxation, such as the linear programming relaxation, becomes infeasible. Analyzing such infeasibilities means to *explain* them and learn from them. An *explanation* of an infeasibility or of a bound update is a set of lower and/or upper bounds of variables that, whenever they occur in that combination, lead to an infeasible state or to that bound update. During conflict analysis the goal is to learn further constraints to detect similar infeasible subproblems earlier. To this end, cuts in a so-called *conflict graph* are computed. This graph consists of the initial explanation, which states a reason for the infeasibility. During the process of the conflict analysis this graph is extended by explanations for bound changes which are part of previous bound explanations. In the end each variable bound in that graph is connected via edges to the elements of its explanation. For a detailed description of the conflict analysis we refer to [1].

When considering propagation algorithms in cumulative scheduling, these explanations are in general not unique and bear a huge potential to optimize the solvers behavior.

The task of an *explanation algorithm* is to *explain* bound changes or infeasible states, i.e., state a set of variable bounds that lead to the deduction, under certain objective criteria. Minimum-size explanations are such a well-suited objective function [4]. Since for each variable multiple bound changes may have been discovered, not only the current bounds can be part of the explanation, but also earlier bound changes. This technique is called *bound-widening* and leads to multiple alternatives per variable that can be reported. Here, theoretical questions about the complexity of computing optimal explanations arise.

4 Results

We show that it is possible to compute in polynomial time minimum-size explanations for bound changes which result from energetic reasoning and edge-finding. In case of time-tabling, we prove that an important special case is already weakly $\mathcal{NP}$-hard. To this end, we establish a relation to *unsplittable flow problems on the path* [3]. We evaluate different heuristic approaches and an exact mixed integer programming approach to explain bound changes derived by that algorithm.

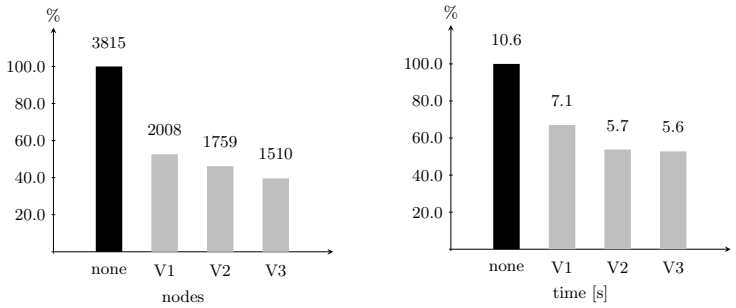

Fig. 1. Comparison of the average number of nodes and average running times for 60 non-trivial and solvable instances from PSPLib `J60` are shown. More sophisticated methods (variants V1 to V3) decrease the number of nodes and the average running times. More detailed results and the different variants can be found in [4].

Using these minimum-size explanations pays off in total compared to using faster but weaker explanation algorithms. In the context of bound-widening, the problems all become $\mathcal{NP}$-hard.

Overall we experience a huge reduction in the number of nodes and in the computation times when using conflict analysis. Figure 1 visualizes the results of [4]. These computational results also reveal the strength of bound-widening techniques, where a widening of minimum-size explanations decreases the running time and the number of nodes additionally.

References

1. Tobias Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, 2007.
2. Timo Berthold, Stefan Heinz, Marco E. Lübbecke, Rolf H. Möhring, and Jens Schulz. A constraint integer programming approach for resource-constrained project scheduling. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010)*, volume 6140 of *LNCS*, pages 313–317. Springer, 2010.
3. Paul Bonsma, Jens Schulz, and Andreas Wiese. A constant factor approximation algorithm for unsplittable flow on paths. *CoRR*, abs/1102.3643, 2011.
4. Stefan Heinz and Jens Schulz. Explanations for the cumulative constraint: An experimental study. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms (SEA 2011)*, volume 6630 of *Lecture Notes in Computer Science*, pages 400–409. Springer, 2011.
5. Andrei Horbach. A boolean satisfiability approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, 181:89–107, 2010.
6. PSPLib. Project Scheduling Problem LIBrary. `http://129.187.106.231/psplib/`.
7. Andreas Schutt, Thibaut Feydy, Peter Stuckey, and Mark Wallace. Explaining the cumulative propagator. *Constraints*, pages 1–33, 2010. 10.1007/s10601-010-9103-2.

Which Mixed Integer Programs could a million CPUs solve?

Thorsten Koch and Yuji Shinano

Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
{koch,shinano}@zib.de

1 Extended Abstract

There is a trend in supercomputing to employ evermore computing cores. Today, the current top 10 machines have on average 150,000 cores and it is likely that a million cores will be available soon. The question arises how to harness these vast computing capabilities to solve new classes of mixed integer programs (MIP).

In the following we will make a few assumptions, namely: Computers having a million cores will use distributed memory. We designate by *processing element* (PE) one shared memory node within this distributed system. One PE might have one or more cores. Furthermore, MIP solvers will be branch-and-cut based, i.e., build a tree with *branch-and-bound* (B&B) nodes, compute linear programming (LP) relaxations of these nodes and derive cutting-planes. See, e.g. [1] for details regarding general MIP solving and [6,5,2] about distributed memory solution techniques.

What are the reasons why MIPs cannot be solved by today's solvers and would using bigger computers help? (See also [3])

1. Genuine bad formulation, e.g. Sudoku with integer variables. Obviously, this is due to improper modeling and the way to solve this is by improving the formulation.
2. Bad dual bounds. Here again improving the formulation would be the solution of choice. Unfortunately, no better or different formulation might be known. In this case using more cycles might allow to solve the instance. Often only improved solvers with better cutting planes, or an improved model will help.
3. Solving LPs is difficult or slow, especially reoptimizing takes long. In the absence of a better way to solve the linear subproblems, this can be resolved by using more cores, once enough open B&B nodes are available.
4. Bad numerical properties. Typically, this needs either an improved model or an improved solver. There are two possibilities to improve the situation by hardware. One is the use of quad-precision floating point arithmetic, which will increase the computational burden. The other is solving numerical difficulties by increased branching instead of solving subproblems. This will increase the search tree.
5. Difficult to find primal solutions. Since more B&B nodes can be evaluated per second and more heuristics can be run, using more cores is likely to be helpful.

6. Large enumeration trees, e.g. due to symmetry. Here the ability to solve more B&B nodes per second clearly helps, even though it is difficult to impossible to predict or estimate the number of B&B nodes that is needed to solve an instance [4]. Basically, a large enumeration tree means that the lower and the upper bound will not converge fast enough. Given that the gap between the bounds is not unreasonably big, as in cases (2) and (5), it is very unclear how many more instances could be solved if 10, 100, 1000 times the number of nodes can be evaluated. Additionally, the maximum number of open B&B nodes that may be encountered during the solution process might get unreasonably big. All non-fathomed nodes have to be stored. It might become necessary to store the majority of those nodes in secondary storage.
7. Too much memory needed for a single B&B node. Here again using bigger computers might help, though the question arises whether it is possible to distribute the processing of one B&B node to more than one shared memory PE. Even how to employ multiple cores to one subproblem is not clear. To solve the LP subproblems, currently, either the Simplex, the Barrier, and variants of sub-gradiant algorithms can be used. We will investigate in detail how much potential and which problems are connected to each of these methods.
8. Nobody knows. In this case computing more nodes should be at least no disadvantage.

In the talk we will concentrate on items (3), (6), and (7).
When using a large number of cores further questions arise:

- Most of the current MIP solvers are designed under the assumption that computing cycles are the main bottleneck. This is not true any more if a million cores are available. What implications does this have to the solver algorithms?
- During the solution of a MIP there are typically several phases, e.g., until the first feasible solution is found, or after an optimal solution has been discovered but not yet proved. Especially in case of (6) the distribution of time between the phases might substantially change. Again the question arises, which implications this has on the MIP solver?
- Typically the time until 1 million active B&B nodes are available is considerable. A similar effect occurs at the end of the computation. How to design these so called, *ramp-up* and *ramp-down* phases?
- Decomposition has always been a topic with much potential that is difficult to realise. Does this change?

We will investigate the above questions in detail during the talk.

References

1. T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

2. D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, USA, 2007.
3. R. E. Bixby. Lectures about LP and MIP solving at Combinatorial Optimization at Work II, 2009.
4. O. Y. Özaltin, B. Hunsaker, and A. J. Schaefer. Predicting the solution time of branch-and-bound algorithms for mixed-integer programs. *INFORMS Journal on Computing*, 2010. DOI: 10.1287/ijoc.1100.0405.
5. Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, and T. Koch. ParaSCIP – a parallel extension of SCIP. Technical Report ZR 10-27, Zuse Institute Berlin, 2010.
6. Y. Xu, T. K. Ralphs, L. Ladányi, and M. J. Saltzman. Computational experience with a software framework for parallel integer programming. *The INFORMS Journal on Computing*, 21:383–397, 2009.

Neuron Constraints to Model Complex Real-World Problems

Michele Lombardi and Michela Milano

DEIS, University of Bologna
{michele.lombardi2,michela.milano}@unibo.it

Late Breaking Abstract

The benefits of combinatorial optimization techniques for the solution of real-world industrial problems is a widely acknowledged evidence, sitting of an ever-growing collection of success stories [1,2,3]. Yet, the application of optimization approaches to many practical domains still encounters active resistance by practitioners. Some examples the authors are familiar with are policy making, environmental engineering and software design for complex hardware platforms.

A considerable part of the issue stems from the difficulty to come up with accurate declarative representations for those domains; this (1) may be due to the presence of factors admitting no obvious numerical assessment (e.g. biodiversity, people preferences), or (2) it may be a result of the interaction of a very large number of elements with known behavior (e.g. complex hardware systems). Whatever the cause, such a representation difficulty has deep practical implications: an over-simplified model may threat the successful application of the most advanced optimization method.

We propose a simple and effective technique to bring remarkably hard-to-handle systems within the reach of Constraint Optimization methods; the goal is achieved by embedding into a combinatorial model a soft-computing paradigm, namely Neural Networks, properly trained before their insertion. Those can provide a reliable approximation for the behavior of complex systems and can learn user preferences or other difficult-to-measure domain elements.

From this perspective, a Neural Network can be thought of as a generic, modular model based on the composition of a single type of non-linear function with vector input. Hence, the integration of such a technique within a combinatorial optimization framework requires from the host technology the capability to handle non-linear expressions; this is provided in CP via global constraints. We therefore proceed by introducing a novel and yet simple class of so-called *Neuron (global) Constraints*, directly modeling a single artificial neuron with a specific activation function, i.e. the expression:

$$f(x) = g\left(\sum_{i=0}^{n-1} w_i x_i - w_n\right) \tag{1}$$

where f is the function representing the artificial neuron and x is the n size input vector; values w_0 to w_n are weights, obtained via a learning stage prior to the

network use; note w_n corresponds to a constant factor. Finally, g is a non-linear function (referred to as *activation function*) squashing the output value in the interval $[-1, 1]$ (or $[0, 1]$). Different types of neurons are distinguished by the adopted type of activation function.

Real valued variables (or finite domain variables via integer approximation) can be associated to the output and to each component of the input vector; hence a neuron constraint has the following signature:

$$\texttt{actfunction}(\texttt{y}, [\texttt{x}], [\texttt{w}])$$

where "`actfunction`" denotes the activation function type — i.e. function g in Equation (1) —, `y` is the output variable, [`x`] is the vector of input variables and [`w`] is the vector of weights. The integration of a fully-fledged (i.e. trained) neural network into a CP model is as straightforward as introducing a Neuron Constraint for each node in the network, connecting input/outputs variables and setting arc weights. Using a global constraint for each single neuron rather than for a whole network provides a fine grain modeling approach, allowing complex networks (even recurrent ones) to be defined with a limited number of basic components, i.e. a constraint for each type of activation function.

As a motivating example for the proposed approach we consider a temperature aware scheduling problem over Multi-Processor Systems on Chip (MPSoC) with Dynamic Frequency Scaling (DFS) capabilities. DFS allows one to slow down one or more processor and let the system cool, so as to become ready to accept more demanding tasks later on. The thermal behavior of an MPSoC device is the result of the interaction of many concurrent factors (heat conduction, heat generation due to processor workload, chip layout...). Moreover, the device quite often embeds low-level thermal controllers making use of DFS during execution to avoid chip overheating: this prevents failures, but may cause unexpected execution delays and result in violations of deadline requirements. Despite the dynamic of the single phenomena is known, the complexity of the overall system makes it very hard to devise a declarative thermal model.

In such a context, a Neural Network can be designed and trained to approximate the behavior of thermal controllers with known input. Specifically, each Neuron Constraint mimics the slow down induced by a single thermal controller and takes into account this approximated behavior into the model. The resulting network can then be embedded in a combinatorial model and used to produce an optimized schedule, avoiding resource over-heating as well as over-usage. Such a temperature friendly schedule improves the system performance and reliability by preventing the unexpected activation of low-level controllers.

References

1. IBM Press Release. Netherlands Railways Realizes Savings of 20 Million Euros a Year With ILOG Optimization Technology. *http://www-03.ibm.com/press/us/en/pressrelease/27076.wss#release*, 2009.

2. INFORMS. Operations Research Success Stories. *http://www.scienceofbetter.org/can_do/success_alpha.php*, 2011.
3. Helmut Simonis. Constraint Application Blog. *http://hsimonis.wordpress.com/*, 2011.

A Constraint Programming Approach for a Batch Processing Problem with Non-identical Job Sizes

Arnaud Malapert[1,3], Christelle Guéret[2], and Louis-Martin Rousseau[3]

[1] École des Mines de Nantes, LINA UMR CNRS 6241
[2] École des Mines de Nantes, YRCCyN UMR CNRS 6597
[3] École Polytechnique de Montréal, CIRRELT
{arnaud.malapert,christelle.gueret}@mines-nantes.fr
louis-martin.rousseau@polymtl.ca

1 Introduction

This paper presents a constraint programming approach for a batch processing machine on which a finite number of jobs of non-identical sizes must be scheduled. A batch processing machine can process several jobs simultaneously. Such machines are encountered in chemical, pharmaceutical, aeronautical and semiconductor wafer industries where an oven, a drier or an autoclave is used during the process. These machines are often bottleneck because of their long processing times. Several papers have been proposed for identical job sizes and due date related performance measures. However, papers on problems with non-identical job sizes, concern mostly completion time related performance measures. For an extensive review on scheduling with batching, we refer the reader to Potts and Kovalyov [4]. Surprisingly, although they have been successfully used to solve various scheduling problems, only one constraint programming approach has been developed for this kind of problem (with sequence dependent setup times and job families).

In this paper, we propose a new constraint programming approach for a problem derived from a real application in aeronautical industry. To our knowledge, only two papers [3,2] concern the resolution of this problem: the minimization of the maximum lateness of a batch processing machine with non-identical job sizes. In these papers, the authors propose a mathematical formulation and a branch-and-price. More formally, the problem can be described as follows. A set J of n jobs and one single batching machine with capacity b are given. Each job j is characterized by an integer triplet (p_j, d_j, s_j), where p_j is its processing time, d_j is its due date, and s_j is its size. The batch processing machine can process several jobs simultaneously as a batch as long as the sum of the sizes of the jobs that are in the batch does not exceed the capacity b of the machine. The processing time of a batch is equal to the longest processing time among the jobs in the batch. The completion time C_j of a job j is the completion time of the batch to which it belongs. The machine and jobs are assumed to be continuously available from time zero onwards, or equivalently, they have equal releases dates. Once the processing

of a batch has been initiated, no job can be removed from or added to the batch. The objective is to minimize the maximum lateness $L_{max} = \max_{1 \leq j \leq n}(C_j - d_j)$. This problem, denoted by $1|p - batch, b < n, non - identical|L_{max}$, is unary NP-hard since Brucker et al. [1] proved that the same problem with identical job sizes is unary NP-hard.

2 Constraint programming approach

The constraint programming formulation relies on the decomposition of the problem into finding an assignment of the jobs to the batches, and then minimizing the maximal lateness of the batches on a single machine.
The problem of assigning the jobs to the batches is equivalent to the *one-dimensional bin packing problem* defined as follows: given n indivisible items (jobs), each of a known non-negative size s_j, and m bins (batches), each of capacity b, can we pack the n items into the m bins such that the sum of the sizes of the items in any bin is not greater than b ? This problem has been shown NP-complete. Our formulation uses a global constraint inspired by Shaw[5].
Then, once the jobs are packed into the batches, the problem of scheduling the batches is equivalent to minimizing the maximal lateness of a set of jobs (batches) on a single machine. This problem, known as $1||L_{max}$, is polynomially solvable: an optimal schedule is obtained by applying Jackson's scheduling rule, also known as the earliest due date (EDD-)rule which schedules the tasks in order of non decreasing due dates. When coping with optimization problems, pruning can be done also on the basis of costs, *i.e.* optimality reasoning. The new constraint `sequenceEDD` uses a relaxation of the problem that yields a lower bound for the objective function to prune portions of the search space. The general idea is to infer primitive constraints on the basis of information on costs. We use optimization components within a global constraint representing a proper relaxation of the problem, which consists of minimizing the maximal lateness of batches on a single machine. The optimization components provide the optimal solution of the relaxed problem, its value and a gradient function computing the cost to be added to the optimal solution for some variable-value assignments. The optimal value of this solution improves the lower bound of the objective function and prunes portions of the search space whose lower bound is bigger than the best solution found so far.

At each node in the search tree, the branching selects a job, and assigns this job to a batch. We chose the variable selection heuristic called *complete decreasing* which packs jobs in order of non-increasing size. We propose a new value selection heuristic called *batch fit* which selects an available bin having the least impact on the schedule of the batches.

3 Experimental results

This section summarizes computational experiments conducted to evaluate our approach. Our algorithm has been tested on randomly generated instances ranging

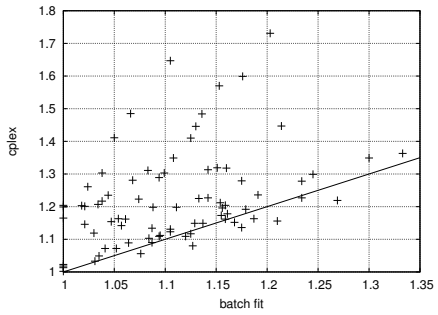

Fig. 1. Comparison to the mathematical formulation.

from 10 to 100 jobs. Our implementation is based on the `choco` solver. All the experiments were conducted on a cluster of Linux machines, each node with 48 GB of RAM and two quad core 2.4 GHz processor. A first set of experiments (not detailed here) show that the use of the constraint `sequenceEDD` as well as the use of the value selection heuristic *batch fit* reduce the computation time and improve solution quality.
Then, our approach is compared to a mathematical formulation solved by `IBM Ilog Cplex` 11.2.1. For small instances ($n \leq 50$), our approach outperforms the mathematical formulation: it solves more instances with solution times that are orders of magnitude lower. Figure 1 compares the quality gap on instances with strictly more than 50 jobs. The quality gap of a solution ub for a given instance is estimated by $(ub+d_{max}) \div (lb+d_{max})$. The time limit of `choco` is 1 hour, whereas it is increased until 12 hours for `Ilog Cplex`. Each point represents one instance and its x coordinate is the quality gap that we obtained, whereas its y coordinate is the quality gap obtained with `Ilog Cplex`. All points located above the line $(x = y)$ are upper bounds improved by the use of our approach. Despite a smaller time limit, the constraint programming approach provides better solution than the mathematical formulation on the vast majority of instances. Furthermore, the difference between the two upper bounds is very tight when the mathematical formulation found the best upper bound, whereas it can be significant when the constraint approach did.

4 Conclusion

We have presented a constraint programming approach to schedule a batch processing machine on which a finite number of jobs of non-identical sizes must be scheduled. This approach exploits an optimization constraint based on a relaxed problem which applies cost based domain filtering rules and is enhanced with a dedicated branching heuristics.
Computational results demonstrate the positive effect of each component and

give better solution with computation times that are orders of magnitude lower than a mathematical formulation based on bin packing and sequencing models.

In further research, we will apply our approach to problems with completion time related measures (C_{max}, $\sum C_j$, $\sum w_j C_j$). In addition, subsequent research topics include the study of parallel batching machines and additional constraints, for instance job release dates that remain incompatible with our approach.

References

1. P. Brucker, A. Gladky, H. Hoogeveen, M. Koyalyov, C. Potts, T. Tautenham, and S. van de Velde. Scheduling a batching machine. *Journal of Scheduling*, 1(1), June 1998.
2. D. Daste, C. Gueret, and C. Lahlou. A Branch-and-Price algorithm to minimize the maximum lateness on a batch processing machine. In *Proceedings of the 11th international workshop on Project Management and Scheduling PMS'08*, pages 64–69, IStanbul Turquie, 2008.
3. D. Daste, C. Gueret, and C. Lahlou. Génération de colonnes pour l'ordonnancement d'une machine à traitement par fournées. In *7ième conférence internationale de modélisation et simulation MOSIM'08*, volume 3, pages 1783–1790, Paris France, 2008.
4. C.N. Potts and M.Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228–249, January 2000.
5. P. Shaw. A constraint for bin packing. In M. Wallace, editor, *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 648–662. Springer, 2004.

Exact Branch-and-price for Fair-share Airline Crew Rostering

Ranga Muhandiramge

Monash University
Melbourne, Australia
ranga.muhandiramge@monash.edu

Abstract. In airlines, assigning staff to flights in generally done in two phases. The first is to group flights into anonymous groups called pairings. The second stage is rostering, that is, to assign these pairings to individual crew members. Traditionally, pairing is concerned mostly with minimizing costs (per diems, hotels, etc.) while rostering is done to optimize the quality of life of, and fairness to, the crew.

We use SCIP to create an exact branch-and-price algorithm for the second, rostering stage. We call our particular model the Fair-share Airline Crew Rostering Problem. A problem instance consists of a set of crew members with preassigned carry-ins and annual leave and a set of pairings that must be assigned. Each pairing and leave shift also has an associated non-negative allowance.

The problem is, firstly, to minimize the number of days contained in unassigned pairings and, secondly, to make the sum of the allowances of the tasks in each crew member's rosterline as close to the given fair-share target as possible; all while respecting the given rostering rules. Examples of rostering rules are a required minimum number of days off after each pairing, dependent upon a pairing's length, and a restriction on to which pairings a pairing of calendar length one day may be prepended.

The problem is formulated as a set partitioning master problem with a pricing subproblem which consists of two Weight Constrained Shortest Path Problems on an acyclic rostering graph. There is one subproblem for rosterlines with an allowance greater than the fair-share target and another for rosterlines with allowance less than or equal the fair-share target.

Our data set, created by our industry partner Constraint Technologies, is constructed to have an ideal optimal solution. That is, one in which each pairing is assigned and the sum of the allowances of each rosterline exactly equal the fair-share target. We also create perturbations of the problem with the same task set but with fewer or more crew to test the robustness of our algorithms.

We examine various combinations of heuristic intialization, exact and heuristic pricing methods and variations on follow-on branching to solve this problem. We also present the results of computational tests from this ongoing work.

Benders Decomposition for the Full-Truckload Pickup-and-Delivery Vehicle Routing Problem

Jenny Nossack and Erwin Pesch

Department of Management Information Sciences, University of Siegen,
Hölderlinstraße 3, 57068 Siegen, Germany
{jenny.nossack,erwin.pesch}@uni-siegen.de

We address a generalization of the well-known vehicle routing problem (VRP) in which the set of customers is divided into pickup and delivery customers and where the former supplies and the latter demands one unit of a given commodity. A fleet of homogeneous vehicles has to be routed such that the supply and the demand of the customers is satisfied while minimizing the total distance traveled. We call this routing problem Full-Truckload Pickup-and-Delivery Vehicle Routing Problem (FTPDVRP). It belongs to the class of many-to-many pickup and delivery problems, where each unit of a pickup customer can be used to accommodate the demand of any delivery customer [2]. The term full-truckload implies unit capacity and unit demand/supply of vehicles and customers, respectively [5]. Besides the exchange of commodities from pickup customers to delivery customers, the depot may additionally fulfill the customers' supply and demand. An extension to the FTPDVRP is the Full-Truckload Pickup-and-Delivery Vehicle Routing Problem with Time Windows (FTPDVRPTW) where customers impose hard time window constraints to denote the time intervals in which commodities have to be picked up/delivered.

One important application of the FTPDVRP arises in the pre- and end-haulage of intermodal container transportation. Macharis and Bontekoning [4] define intermodal container transportation as the movement of containers by two or more transportation modes (rail, maritime, and road) in a single transport chain. The change of modes is thereby performed at bi- and tri-modal terminals without handling the freight itself. The route of intermodal transport is namely subdivided into the pre-, main-, and end-haulage, denoting the route segments from customer to terminal, terminal to terminal, and terminal to customer, respectively. The fundamental transportation assignments arising in the pre- and end-haulage are the movements of containers between customers (shippers or receivers) and container terminals and vice versa. These transportation assignments are typically carried out by a truck company to enable house-to-house transports. In some applications it is further the case that the containers are owned by the truck company. Here, a shipper customer demands an empty container and a receiver customer supplies an empty container. Problems of this type can be modeled as a FTPDVRP.

We model the FTPDVRP as an integrated nonlinear programming problem that simultaneously solves an assignment and a routing problem, linked via coupling constraints. It is part of the assignment problem to fulfill the demand and the supply of the customers by determining where to deliver the commodities

that are collected from the pickup customers and where to pick up the commodities that are required by the delivery customers. The routing problem evaluates in which order and by which vehicle the customers are visited by taking into account the decisions of the assignment problem. If the variables of the routing problem are fixed, the remaining assignment problem is easy to solve. This particular property points to a (generalized) Benders decomposition approach for solving instances of the FTPDVRP. Hence, we solve the nonlinear programming problem of the FTPDVRP by the generalized [3] and its linearization by the classical Benders decomposition [1] approach. For the FTPDVRPTW we suggest a two-phase heuristic that solves an assignment problem in the first phase and constructs vehicle routes in the second phase. The performance of the algorithms were tested on randomly generated instances.

References

1. Benders, J.F.: Partitioning Procedures for Solving Mixed-Variables Programming Problems. Numerische Mathematik 4, 238-252 (1962)
2. Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., Laporte, G.: Static Pickup and Delivery Problems: A Classification Scheme and Survey. Top 15, 1-31 (2007)
3. Geoffrion, A.M.: Generalized Benders Decomposition. Journal of Optimization Theory and Applications 10, 237-260 (1972)
4. Macharis, C., Bontekoning, Y.M.: Opportunities for OR in Intermodal Freight Transport Research: A Review. European Journal of Operational Research 153, 400-416 (2004)
5. Parragh, S.N., Doerner, K.F., Hartl, R.F.: A Survey on Pickup and Delivery Problems. Journal für Betriebswirtschaft 58, 81-117 (2008)

Multimodal Home Healthcare Scheduling using a novel CP–VND–DP Approach

Andrea Rendl, Matthias Prandtstetter, and Jakob Puchinger

Mobility Department, Austrian Institute of Technology, Vienna, Austria
{andrea.rendl,matthias.prandtstetter,jakob.puchinger}@ait.ac.at

1 Overview

Healthcare services, especially home healthcare services (HHC), are of great significance in today's western world, where the average age is steadily increasing. HHC services are particularly popular, since patients prefer being nursed at home than at retirement homes. However, finding a good schedule is challenging:

- Nurses need to arrive at the patients' homes in certain **time windows**
- Each service requires a **qualification** that the nurse must hold (e.g. a nurse for cleaning may not perform a medical service)
- The schedule should meet **preferences** of patients, nurses and employer
- All **legal and contractual issues** (such as sufficient resting periods) should be satisfied

We consider a real-world setting, where the objective is to find a schedule with *minimal travel time* (to reduce operational costs), such that *all* patients are assigned a nurse, satisfying as many side constraints from the above as possible. Our goal is to construct a flexible framework that generates efficient nurse schedules for different home care companies.

Real-world HHC problems are challenging for many reasons. First, the HHC problem constitutes a combination of two NP-hard problems: vehicle routing with time windows [4,5] and nurse rostering [6]. Therefore, we need to employ a heuristic approach since exact solving methods can typically only tackle small instances. Second, real-world instances include many side constraints that vary greatly between different service providers (e.g. different nurse contracts). Hence we require a *flexible* solving architecture, where constraints can be easily added/removed. Third, to provide accurate schedules, it is crucial to use accurate travel time estimations, regarding *different modes of transport*. Therefore, we include travel time estimations from a large set of historical data for different transport modes (car, public transport, bike, foot) into our system.

2 A Hybrid Approach for Solving the HHC Problem

We propose a hybrid meta-heuristic for the HHC problem where constraints can be easily added and/or removed, and different transport modes are considered. The solution approach consists of three parts: (1) an initialization step, (2) an improvement phase and (3) a constraint checking and subproblem solution phase. Three core components are used during those steps: a *constraint programming* (CP) [11] formulation, a *variable neighborhood descent* (VND) [8] improvement phase and a *dynamic programming* (DP) [2] method.

1. Initialization Step First, an initial solution is generated using a CP approach, where we use an extension of the vehicle routing problem model from [11]. The advantages of using CP are threefold: first, the CP model is *flexible*, since constraints can be easily added and/or removed from the model. Second, CP is efficient in finding good first solutions, if adequate search strategies are chosen. Third, the CP model can be reused to check if a modified solution still satisfies all constraints.

2. Solution Improvement Second, the initial solution is improved by VND, a fast local search method, where various neighborhoods, which are generated by 'moves' such as '*swap-nurses*', are systematically searched for better solutions. Candidate solutions are verified by reloading the store of the CP model. This approach guarantees that the system is kept flexible, yet accurate, since constraints can be easily added/removed without any major implementation tasks.

3. Tour Planning Phase In a third step, we improve the tours of the schedule: as soon as all patients are assigned to a nurse, planning tours can be considered an independent subproblem, *a TSP with time windows*. First, we reduce the TSP with time windows to a generalized TSP by applying the algorithm from Albiach et al [1]. Second, we solve the TSP using an extended DP-approach based on [9]. The main benefit of using DP is that we obtain an *exact* solution in little time due to the instance dimensions (limited number of nodes per tour).

Multimodality Nurses typically travel by (a combination of) different transport modes (car, bike, foot, public transport) which has an effect on the overall travel time. Currently, the transport mode of each nurse is given *in advance* and therefore constitutes a feature (such as *qualification*) that affects travel time. Thus, multimodality results in additional side constraints. The travel times are computed from time-dependent historical data for each transport mode. In future work, we want to *select* the best transport mode for each nurse.

3 Related Work

The main difference to other approaches lies within our objective to provide a *flexible* framework to tackle real-world HHC problems of different flavor. Hence, our hybrid setup is tailored to this requirement. Other meta-heuristical approaches [7,3,12,10] use different setups and do not consider multimodality.

4 Discussion and Conclusions

In this abstract, we propose a new hybrid meta-heuristic to solve real-world HHC instances in a flexible framework for different homecare companies. The contributions of this work are twofold: first, we motivate a *flexible*, novel, hybrid approach to tackle HHC problems. Second, we introduce the issue of *multimodality* in HHC: in our current setting, we investigate the case where nurses select their preferred mode of transport in advance. However, we are interested in the extension of this problem, where we optimize the transport mode for each nurse wrt different objectives (minimization of travel time, travel costs, etc). Can we obtain better nurse schedules by additionally choosing the mode of transport for each nurse?

Acknowledgments

This work is part of the project CareLog, partially funded by the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) within the strategic programme I2VSplus under grant 826153. The authors thankfully acknowledge the CareLog project partners Verkehrsverbund Ost-Region GmbH (ITS Vienna Region), Sozial Global AG, and ilogs mobile software GmbH.

References

1. Albiach, J., Sanchis, J.M., Soler, D.: An asymmetric tsp with time windows and with time-dependent travel times and costs: An exact solution through a graph transformation. European Journal of Operational Research 189(3), 789–802 (2008)
2. Bellman, R.: Dynamic programming. Princeton University Press (1957)
3. Bertels, S., Fahle, T.: A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem. Computers & Operations Research 33, 2866–2890 (October 2006)
4. Bräysy, O., Gendreau, M.: Vehicle routing problem with time windows, part i: Route construction and local search algorithms. Transportation Science 39(1), 104–118 (2005)
5. Bräysy, O., Gendreau, M.: Vehicle routing problem with time windows, part ii: Metaheuristics. Transportation Science 39(1), 119–139 (2005)
6. Burke, E.K., De Causmaecker, P., Berghe, G.V., Van Landeghem, H.: The state of the art of nurse rostering. Journal of Scheduling 7, 441–499 (November 2004), http://portal.acm.org/citation.cfm?id=1029473.1029477
7. Eveborn, P., Flisberg, P., Rnnqvist, M.: Laps Care–an operational system for staff planning of home care. European Journal of Operational Research 171(3), 962–976 (2006)
8. Hansen, P., Mladenović, N.: Variable neighborhood search. In: Glover, F.W., Kochenberger, G.A. (eds.) Handbook of Metaheuristics, pp. 145–184. Kluwer Academic Publisher, New York (2003)
9. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics 10(1), 196–210 (1962)
10. Rasmussen, M.S., Justesen, T., Dohn, A., Larsen, J.: The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. Tech. Rep. 11-2010, DTU Management Engineering (May 2010)
11. Rossi, F., Beek, P.v., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA (2006)
12. Steeg, J., Schröder, M.: A hybrid approach to solve the periodic home health care problem. In: Kalcsics, J., Nickel, S. (eds.) OR. pp. 297–302. Springer (2007)

Search Combinators

Tom Schrijvers[1], Guido Tack[2], Pieter Wuille[2],
Horst Samulowitz[3], and Peter J. Stuckey[4]

[1] Universiteit Gent, Belgium
[2] Katholieke Universiteit Leuven, Belgium
[3] IBM Research, USA
[4] National ICT Australia (NICTA) and University of Melbourne, Victoria, Australia

This work introduces search combinators, an approach to modeling search in constraint solvers that enables users and system developers to quickly design complex efficient search heuristics.

Search heuristics often make all the difference between effectively solving a combinatorial problem and utter failure. Heuristic enable a search algorithm to become efficient for a variety of reasons, e.g., incorporation of domain knowledge, or randomization to avoid heavy tailed runtimes. Hence, the ability to swiftly design search strategies that are tailored towards a problem domain is essential to performance improvement.

While we focus on systematic tree search in the area of Constraint Programming (CP), we believe that the results can be adapted to other search-driven areas in the field of Artificial Intelligence (AI) and related areas such as Operations Research (OR) (e.g., for Mixed Integer Programming (MIP) solvers). In fact, we also believe that our approach anticipates the challenges that will surface when one needs to control search in the context of *hybrid systems* that are composed of a variety of solvers (e.g., a mix of CP and MIP). In CP, much attention has been devoted to facilitating the modeling of combinatorial problems. A range of high-level modeling languages, such as Zinc [2] and OPL [4], enable quick development and exploration of problem models. However, we see very little support on the side of formulating accompanying search heuristics. Either the design of search is restricted to a small set of predefined heuristics (e.g., MiniZinc [3]), or it is based on a low-level general-purpose programming language (e.g., Comet [5]). The former is clearly too confining, while the latter leaves much to be desired in terms of productivity, since implementing a search strategy quickly becomes a non-negligible effort. This also explains why the set of available heuristics is typically small: it takes a lot of time for CP system developers to implement heuristics, too – time they would much rather spend otherwise improving their system.

In this work we show how to resolve this stand-off between solver developers and users with respect to a high-level search language.

For the user, we provide a compositional approach for expressing complex search heuristics based on an (extensible) set of primitive combinators. Even if the users are only provided with a small set of combinators, they can already express a vast range of combinations. Moreover, programming search in terms of combinators is far more productive than resorting to a low-level language.

The following heuristic briefly demonstrates the conciseness of our search combinators. This search heuristic can be used to solve radiotherapy treatment planning problems [1]. The heuristic minimizes a variable *obj* using branch-and-bound (bab), first searching the variables N, and then verifying the solution by partitioning the problem along the row_i variables, one row at a time. Failure on one row must be caused by the search on the variables in N, and consequently search never backtracks into other rows (exh_once). This is a good example of integrating domain knowledge in search:

```
bab(obj, and([int_search(N, input_order, bisect_low),
              exh_once(int_search(row_1, input_order, bisect_low)),
              ...,
              exh_once(int_search(row_n, input_order, bisect_low))]))
```

Here we assume that a basic search construct like the following is available:

$$s = \mathsf{int\_search}(vars,\ \textit{var-select},\ \textit{value-select})$$

which specifies a systematic search over the variables *vars*, applying *var-select* and *value-select* as variable- and value-selection strategies respectively. Another primitive combinator is and, with the obvious meaning. The remainders, bab and exh_once, are themselves defined in terms of other primitive combinators.

For the system developer, we show how to design and implement modular combinators. Developers do not have to cater explicitly for all possible combinator combinations. Small implementation efforts result in providing the user with a lot of expressive power. Moreover, the cost of adding one more combinator is small, yet the return in terms of additional expressiveness can be quite large.

In summary, the tough technical challenge we face is to bridge the gap between conceptually simple specification language (high-level, purely functional and naturally compositional) and efficient implementation (typically low-level, imperative and highly non-modular). We overcome this challenge with a systematic approach that disentangles different primitive concepts into separate modular components, search combinators, that interact through a *message protocol*. This protocol dictates how the combinators should collaborate to process a node in the search tree. Overall search then consists of a queue of unprocessed nodes that are fed one by one to the combinators, which in turn may produce new (child) nodes for the queue. The message-based combinator approach lends itself well to different implementation strategies. We have developed two diametrically opposed approaches for the Gecode C++ library: *dynamic composition* (interpretation) and *static composition* (compilation). Experimental evaluation shows that both implementation approaches have competitive performance and match the performance of the native implementation of the same search heuristics in Gecode.

References

1. Baatar, D., Boland, N., Brand, S., Stuckey, P.: CP and IP approaches to cancer radiotherapy delivery optimization. Constraints (2011)

2. Marriott, K., Nethercote, N., Rafeh, R., Stuckey, P., Garcia de la Banda, M., Wallace, M.: The design of the Zinc modelling language. Constraints 13(3), 229–267 (2008)
3. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: Bessire, C. (ed.) CP. LNCS, vol. 4741, pp. 529–543. Springer (2007)
4. Van Hentenryck, P.: The OPL optimization programming language. MIT Press (1999)
5. Van Hentenryck, P., Michel, L.: Constraint-Based Local Search. MIT Press (2005)

Towards a Characterization of Adaptiveness for Constraint Programming Search Design

Thiago Serra

Instituto de Matemática e Estatística, Universidade de São Paulo, Brazil
tserra@ime.usp.br

Objective and Motivation The aim of this work is to propose a new characterization of *adaptive search methods* for Constraint Programming (CP). It is motivated by the large number of previous works on the topic, which altogether indicate certain patterns for search design decisions. Most of these previous approaches share several unexplored similarities, usually disregarded by the lack of a standard categorization. Hence, a review on the current classification scheme could potentially facilitate the use and future development of CP adaptive search, helping practitioners to identify the search procedures that best suit their needs.

Background and Related Work The recent success of Constraint Programming (CP) can be largely attributed to its expressive modeling capabilities. Nevertheless, this increased modeling power is only meaningful if associated with a robust search algorithm [3]. Besides, search automation is regarded as an important step towards broadening the community of CP users [4]. In particular, many researchers have been tackling constraint satisfaction problems (CSPs) through adaptive mechanisms that guide the choice of the most appropriate procedure for each occasion.

Adaptive strategies modify the search behavior based on a mapping of static and dynamic information onto performance metrics. This information is derived through an inference process that takes into account the problem model, available instances, and particular algorithmic structures. It requires learning how the subtleties of each case explains the observed performance variations so that search is adapted accordingly. Such learning task is known as *algorithm selection* in the field of *meta-learning*, which has been recently a subject on many CSP approaches [5]. A formal taxonomy to classify these adaptive search mechanisms according to their design has been first proposed in [2]. It defines an hierarchical scheme consisting of three layers, described as follows: (i) when search modifications occur, (ii) which information induces them, and (iii) how search is being modified. Furthermore, an introduction to the topic and its design issues with a focus on stochastic local search algorithms is given in [1].

Approach This work intends to deepen on the characterization of decision classes for designing adaptive search procedures. It decouples the layered architecture presented in [2] into independent functional patterns, emphasizing the usual

CP structure and search design particularities. Each of those patterns is briefly described as follows.

Learning Patterns The adaptation of search relies on some learning process whose outcome is the information that will endorse its choices. There are three variations in timing and frequency of such process within the overall system execution. *Batch:* Search is tuned once and prior to use through the analysis of *what-if* scenarios over a set of initial instances, which are expected to be a meaningful sample of the real case usage. *Online:* At each run, search is tuned from scratch according to information gathered during its execution. *Offline:* Between consecutive runs, search is progressively tuned according to tests on data from previous instances.

Decision Patterns The outlining of search procedures is made through the degrees of freedom purposely left on search design. They can be placed at three abstraction levels, which varies from selection of algorithms down to their inner assembly. *Algorithm Portfolio:* Processing power is split among procedures and each of them is expected to explore the search space in a different order. *Algorithm Selector:* A single procedure is selected to perform the entire search, what is also called a "winner take all". *Algorithmic Framework:* Search is decomposed into components with distinct responsibilities, some of which with more than one available strategy. In CP, such framework usually corresponds to a combination of propagation, retraction and branching heuristics.

Discrimination Patterns The way in which problem data is used to learn and to outline the search creates explanatory theories for such choices. They differ by extension of details and specificity of domain in three ways. *Atomic:* The problem is used as a basic unit upon which search performance must be improved, ignoring inner details. *General Profile:* General properties of search algorithms and CSPs are used to characterize performance variations. *Domain-Specific:* Specific problem structures are used to control the search, such as global constraints.

References

1. Battiti, R., Brunato, M., Mascia, F.: Reactive Search and Intelligent Optimization. Springer, New York (2008)
2. Hamadi, Y., Monfroy, E., Saubion, F.: What is Autonomous Search? Technical Report MSR-TR-2008-80, Microsoft Research, Cambridge (2008)
3. Lustig, I.J., Puget, J. -F.: Program Does Not Equal Program: Constraint Programming and Its Relationship to Mathematical Programming. Interfaces, 31, 6 (2001)
4. Puget, J. -F.: Constraint Programming Next Challenge: Simplicity of Use. In: Proceedings of CP 2004, pp. 5-8, Toronto (2004)
5. Smith-Miles, K. A.: Cross-Disciplinary Perspectives on Meta-Learning for Algorithm Selection. ACM Comput. Surv., 41, 1, Article 5 (2008)

Using column generation to solve the edge coloring problem

J.M. van den Akker, J.A. Hoogeveen, and W. Lauret

Department for Information and Computing Sciences
Utrecht University
Princetonplein 5, 3584 CC Utrecht, The Netherlands
{J.M.vandenAkker,J.A.Hoogeveen}@cs.uu.nl
wouter.lauret@gmail.com

Introduction

The edge coloring problem is defined as follows. Given an undirected graph $G = (V, E)$, color the edges of the graph with a minimum number of colors such that all edges containing the same vertex get a different color.

Obviously, since each vertex incident to a given vertex v must get a different color, we need at least $d(v)$ colors, where $d(v)$ is equal to the degree of vertex v. Hence, we find that the maximum degree $d_{\max}$ is a lower bound to the number of colors needed in a feasible coloring. Vizing (1964) showed that, either $d_{\max}$ colors are sufficient, or we need $d_{\max} + 1$ colors. Holyer (1981) showed nonetheless that the edge coloring problem is $\mathcal{NP}$-hard.

Our solution method

We want to attack the edge coloring problem using ILP-techniques. Our formulation is based on partitioning the graph into as few as possible subsets of edges that get the same color. In this way, we circumvent the symmetry problems that would arise if we would assign colors to individual edges. It is well-known that a one-colorable subset corresponds to a matching. Note that we cannot limit ourselves to maximum cardinality matchings, as can be seen from the simple example of a path consisting of three edges.

Suppose that we know the set S containing all matchings. We introduce for each matching $s \in S$ a binary decision variable x_s. We define a_{es} as a binary parameter that indicates whether matching s contains edge $e \in E$ ($a_{es} = 1$ if it does, and $a_{es} = 0$, otherwise). Then the ILP-formulation becomes

$$\text{minimize} \quad \sum_{s \in S} x_s$$

subject to

$$\sum_{s \in S} a_{es} x_s \geq 1, \text{ for each } e \in E.$$

$$x_s \in \{0, 1\}, \text{ for each } s \in S.$$

The first constraint enforces that each edge should belong to at least one of the selected matchings; if an edge belongs to more than one, then we remove it from all chosen matchings but one.

There are two well-known problems with this approach: we do not know all matchings, and if we would, then solving the ILP-formulation would last for ever. Therefore, we take the LP-relaxation, which is obtained by replacing the constraint that x_s should be binary by the constraint that x_s should be nonnegative (we do not need to enforce the upper bound of 1, since this is clearly sub-optimal). We solve the LP-relaxation using column generation. To start the procedure, we initialize with all matchings consisting of a single edge. Suppose that we have found the optimal solution of the LP-relaxation for a given set of matchings. Let π_e be the dual multiplier for the constraint corresponding to edge $e \in E$. The reduced cost of a matching s is then equal to

$$c'_s = 1 - \sum_{e \in E} a_{es} \pi_e.$$

Hence, our pricing problem, in which we search for a column with minimum reduced cost, becomes a maximum weight matching problem; if the weight of the matching is more than one, then we add it; if it is no more than one, then we have solved the LP-relaxation to optimality. The maximum weight matching problem is solvable in $O(nm + n^2 \log n)$ time using Gabow's (1976) implementation of Edmonds's (1965) algorithm, where m and n correspond to the number of edges and vertices in the graph, respectively.

Since we only want to distinguish between the values $d_{\max}$ and $d_{\max} + 1$, and since the LP-relaxation gives a lower bound to the value of the ILP, we don't care about the exact optimum, if we know that it exceeds $d_{\max}$. To prevent that we have to wait until termination of the column generation algorithm, we use an intermediate lower bound, which has been described by Van den Akker, Hoogeveen, and van Kempen (2006). Let c^* denote the value of the maximum weight matching. Then

$$\frac{\sum_{e \in E} \pi_e}{c^*}$$

is a lower bound on the outcome value of the LP-relaxation. If this value exceeds $d_{\max}$, then we know that we need $d_{\max} + 1$ colors.

If we find that the optimum to the LP-relaxation is equal to $d_{\max}$, then we solve the ILP-formulation with the set of columns that we have determined when solving the LP-relaxation. If we find a solution using $d_{\max}$ colors then, then we have solved the problem. If we do not succeed in this, then the problem is currently undecided, and we may proceed using branch-and-price (see Barnhart et al., 1998).

Computational results

We have tested our approach to some benchmark instances available in the literature like the DIMACS graphs. Unfortunately, these all are colorable using

$d_{\max}$ colors, which is discovered when we solve the ILP-formulation with the set of columns determined in the column generation phase. We will apply perturbations to these graphs, such that we may need $d_{\max} + 1$ colors, to test whether the LP-relaxation is able to decide these graphs.

References

1. J.M. van den Akker, J.A. Hoogeveen, and J.W. van Kempen (2006). Parallel Machine Scheduling Through Column Generation: Minimax Objectives. In Y. Azar and T. Erlebach (eds.), *Algorithms ESA 2006, Lecture Notes in Computer Science, volume 4168*, Springer-Verlag Berlin Heidelberg, Volume, 648–659.
2. C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance (1998). Branch-and-price: column generation for solving huge integer programs. *Operations Research 46*, 316–329.
3. J. Edmonds (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics 17*, 449–467.
4. H.N. Gabow (1976). An efficient implementation of Edmonds's maximum matching on graphs. *Journal on the ACM 23*, 221–234.
5. I. Holyer (1981). The NP-completeness of edge-coloring. *SIAM Journal on Computing 10*, 718–720.
6. V.G. Vizing (1964). On an estimate of the chromatics class of a p-graph. *Diskret. Analiz. 3*, 23–30.

The Aimms Interface to Constraint Programming

Willem-Jan van Hoeve[1], Marcel Hunting[2], and Chris Kuip[2]

[1] Tepper School of Business, Carnegie Mellon University
[2] Paragon Decision Technology

Abstract. We present an extension of the modeling system Aimms to handle constraint programming problems. Our goal is to provide a more accessible interface to CP technology than current systems offer. We first present basic CP modeling constructs that can be realized with minimum changes to the existing syntax. We then discuss the handling of global constraints. Lastly, we present our extensions to modeling scheduling problems, based on the now-classical representation as activities and resources. An important benefit of the Aimms interface to CP is the ease with which hybrid CP/OR solution methods can be developed.

1 Introduction

From its inception, the academic field of Constraint Programming (CP) has been coupled with well-engineered solvers that were applied to challenging, often large-scale, industrial problems (e.g., CHIP, Cosytec, ILOG Solver, SICStus, Eclipse, Choco, Kalis, Gecode, Comet). Most, if not all, of these solvers require a considerable amount of training before they can be used by an engineering student or Operations Research practitioner, let alone an MBA student. One reason for this is that each of these solvers has its own specific modeling language, for example in the style of Prolog or C++. Another reason is that CP requires a different modeling style than mathematical programming, to the extent that even a dynamic search strategy can be modeled in most systems. Finally, even though most available solvers share the most important characteristics of CP, each comes with its own library of global constraints, associated filtering algorithms, modeling concepts (e.g., resource constrained scheduling, set variables, local search), and specialized search techniques. Consequently, a problem may be modeled and solved in entirely different ways by different solvers. All of this limits the accessibility of CP to typical Operations Research practitioners.

One of the main goals of the project described in this abstract is to make CP more accessible to a wider range of practitioners, from industrial OR experts to MBA students. In order to achieve this, we have developed a Constraint Programming extension of the modeling system Aimms, keeping the following requirements and targets in mind. First, we would like the difference between standard mathematical programming models and CP models to be minimal for a user, and in particular for an Aimms user. Second, we must offer the most

powerful CP technology that is available (e.g., access to advanced algorithms for resource-constrained scheduling), as well as all common global constraints. Third, the system should be designed for solving hard industrial problems. Note that these targets can be conflicting, e.g., the support of global constraints for their filtering power does not align with standard mathematical programming terminology.

2 Background and Related Work

Several other industrial and academic modeling languages/systems have been developed to make the use of CP and hybrid methods more accessible. Academic systems focusing on the integration of solvers include Zinc [9] and SIMPL [13]. Industrial systems include IBM ILOG CPLEX Optimization Studio (with the OPL language [11]), Fico Xpress Optimization Studio (with the Mosel language [4]), and Comet [12]. Even though these systems are powerful and provide advanced interfaces, they mostly appeal to specialized developers, and suffer from the aforementioned shortcomings with respect to accessibility to non-specialists, in our opinion. Our goal is to contribute to these developments by focusing specifically at a solver-independent, intuitive graphical modeling interface, as offered by the AIMMS system, in order to attract non-experts to use CP technology.

The modeling language underlying AIMMS [2,10] was originally developed in a similar spirit as GAMS [3] or AMPL [6]. Similar to those systems, it is based on an algebraic syntax and offers access to (at least) ILP, QP, and NLP technology. The main difference with these other languages, however, is that model development in AIMMS revolves around a graphical modeling interface depicting the hierarchical structure in a model formulation. This enables the user to formulate a problem in an intuitive and naturally decomposed manner. A second important feature of AIMMS is that it offers user-developed 'pages', that can be used to graphically depict solutions and even to build entire end-user applications that can be deployed in practice.

3 Contributions

The AIMMS extension to CP supports all common global constraints, ranging from `AllDifferent` to `Sequence` and `BinPacking`. In addition, it supports advanced scheduling concepts based on the classical representation using 'activities' and 'resources' [1], as well as specialized global scheduling constraints as introduced recently in [8]. Interestingly, many modeling concepts from CP were already present in the existing AIMMS syntax, albeit restricted to non-variable identifiers. Namely, AIMMS already offers all standard arithmetic, logical, and set related operators. This allowed us to provide CP functionality with minimal changes to the existing syntax in many cases. Finally, we have built an interface to the CP solvers IBM ILOG CP Optimizer and Gecode. We refer to [7] for more details.

In summary, the most important contribution of our extension to the practice of CP is that less-experienced users can access CP technology in a more intuitive

way, and furthermore make use of all standard features of AIMMS including its advanced graphical interface. In addition, we remark that the related modeling systems GAMS and AMPL do not support CP technology,[1] while OPL, Mosel, and Comet do not provide access to NLP solvers such as CONOPT, KNITRO, IPOPT and LGO. With the addition of our CP interface, AIMMS is currently the only industrial-strength optimization modeling system that provides access to LP, MIP, QP, NLP, and CP solvers, which makes it possible to build integrated models combining all these technologies.

References

1. P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, 2001.
2. J. Bisschop and R. Entriken. *AIMMS: The modeling system*. Paragon Decision Technology, 1993.
3. J. Bisschop and A. Meeraus. On the development of a general algebraic modeling system in a strategic planning environment. *Mathematical Programming Study*, pages 1–29, 1982.
4. Y. Colombani and S. Heipcke. Mosel: An Extensible Environment for Modeling and Programming Solutions. In *Proceedings of the Fourth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems (CPAIOR)*, 2002.
5. R. Fourer and D.M. Gay. Extending an algebraic modeling language to support constraint programming. *INFORMS Journal on Computing*, 14:322–344, 2002.
6. R. Fourer, D.M. Gay, and B.W. Kernighan. AMPL: A Modeling Language for Mathematical Programming. *Management Science*, 36:519–554, 1990.
7. W.J. van Hoeve, M. Hunting, and C. Kuip. Constraint programming. In *AIMMS 3.12: The Language Reference*, chapter 21. Paragon Decision Technology, 2011.
8. P. Laborie. IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems. In *Proceedings of CPAIOR*, volume 5547 of *LNCS*, pages 148–162. Springer, 2009.
9. K. Marriott, N. Nethercote, R. Rafeh, P.J. Stuckey, M. Garcia De La Banda, and M. Wallace. The design of the Zinc modelling language. *Constraints*, 13(3):229–267, 2008.
10. M. Roelofs and J.J. Bisschop. *AIMMS 3.11: The Language Reference*. Paragon Decision Technology, 2010. http://www.aimms.com/downloads/manuals/language-reference.
11. P. Van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999. With contributions by I. Lustig, L. Michel, and J.-F. Puget.
12. P. Van Hentenryck and L. Michel. *Constraint-Based Local Search*. The MIT Press, 2005.
13. T. Yunes, I.D. Aron, and J.N. Hooker. An integrated solver for optimization problems. *Operations Research*, 58(2):342–356, 2010.

[1] We note that the AMPL language extensions to CP, as described in [5], have not yet been implemented.

Solving the no-wait job shop problem: an ILP and CP approach

H.M. Vermeulen, J.A. Hoogeveen, and J.M. van den Akker

Department for Information and Computing Sciences
Utrecht University
Princetonplein 5, 3584 CC Utrecht, The Netherlands
vermeulen99@yahoo.com
{J.A.Hoogeveen,J.M.vandenAkker}@cs.uu.nl

Introduction

In the traditional job shop scheduling problem, we have a set of n jobs that have to be executed by a set of m machines. Each job consists of a set of operations, and for each job it is given by which machine it has to be executed and how long this takes. Moreover, there is a strict order between these operations, and the kth operation in job j can only start when the $(k-1)$th operation of job j has been completed. The objective is to minimize the time by which each job has been completed, which is called the makespan. The job shop problem has received a lot of attention during the last fifty years. It is not only $\mathcal{NP}$-hard, but also it is hard to solve the problem to optimality for instances with more than 20 jobs and machines. These enumerative algorithms were traditionally based on applying branch-and-bound to the disjunctive graph model, but in the last decade we have seen attempts to solve these problems through constraint satisfaction (see for instance Baptiste et al., 2001).

The no-wait job shop problem is a variant of the traditional job shop scheduling problem, where the time between operations of the same job is fixed. These time differences between the completion time of the $(k-1)$th operation and the kth operation do not have to be zero; these can even be negative, implying that the operations overlap in their execution. As a result, we know the start times of all operations, as soon as the start time of the job gets fixed. Since we assume that the machines can handle only one job at a time, we must choose the start times of the jobs such that no two operations that must be executed by the same machine overlap in their execution. Hence, we can compute for each pair of jobs the values of the differences of their start times such that these two jobs do not collide in their execution. Therefore, our problem boils down to finding for each job its start time, such that the last job gets completed as soon as possible, and such that each start-time difference must fall into a fixed set of intervals, which can be easily calculated. This deceivingly simple-looking problem turns out to be quite hard to solve within a reasonable amount of time.

Our solution methods

We have developed exact solution methods using Integer Linear Programming (ILP) and a combination of binary search with Constraint Programming (CP) to find an optimal makespan. Below we shortly summarize the methods we have applied, and we end with some conclusions.

We have applied several ILP formulations. The first ones we tested were based on the time-indexed formulation, where we have a binary variable x_{jt} for each job j and possible start time t. Modeling the forbidden start-time differences requires many constraints, which resulted in very large run times, even for instances with 5 jobs. Next, we applied models with variables S_j to model the start times. We further introduced binary variables x_{ijk} to model that $S_j - S_i$ should fall in the kth interval of the allowed set of start time differences. We have strengthened this formulation by adding constraints based on the allowed start time differences for triples of jobs, which can be determined using a three-job propagator we used in the CP methods. Finally, we attempted models in which the non-overlap constraints were modeled using a disjunctive formulation. It turned out in our experiments that the disjunctive formulation where we explicitly used the allowed set of start time differences worked best.

To start CP, we put an upper bound on the makespan and check whether there exists a feasible solution; the optimum can then be determined through binary search. This upper bound can easily be translated into a deadline on the start time of each job. The variables we used were the starting times of all jobs. We used the so-called two-job propagator to restrict the domains as far as possible. The two-job propagator modifies the domain of the starting time of job j given the domains of the starting time of job i and the allowed start time differences. A nice property of the two-job propagator is that it maintains the so-called arc-consistency. The two-job propagator dominates the so-called *Time-Table* and *Disjunctive* propagators, which are very successful in solving the standard job-shop scheduling problem. Another start time variable based propagator that we employed is the *three-job difference propagator*, which works on the domains of the start time differences; this propagator does not maintain full path consistency.

We derive two simple but effective propagators which maintain full arc consistency and partial path-consistency on the start-time variables.

We performed extensive experiments using different ILP formulations, lower bounds, constraint propagation strategies and branching strategies. We show that the edge-finding and not-first, not-last propagators used in solving the traditional job shop problem are ineffective in the presence of no-wait constraints. We also show that it is hard to combine ILP and CP to give a better algorithm than both separately.

Our best CP strategy outperformed our best ILP method by a factor of 5 to 50. For medium-sized problem instances of 10 jobs the performance of our best CP strategy is comparable to that of the best exact algorithm (Van den Broek, 2009) for the no-wait job shop problem, which uses branch and bound, but its running time increases much faster than that of the branch-and-bound

method as the number of jobs increases. However, at a fixed number of jobs, our CP method is better in handling an increase in the number of machines and operations per job by exploiting the fact that overlap between jobs on any machine can be detected by only using the start-time difference. We are hopeful that in the future this can be incorporated into a branch-and-bound algorithm with a much better performance for a larger number of machines and operations per job.

References

1. P. Baptiste, C. le Pape, and W. Nuijten (2001). *Constraint-Based Scheduling*, Kluwer.
2. J.J.J. van den Broek (2009). *MIP-based approaches for complex planning problems*. PhD Thesis, Eindhoven University of Technology, The Netherlands.