The 18^{th} International Conference on Principles and Practice of Constraint Programming (CP'12)

Doctoral Programme Proceedings

8 - 12 October 2012
Québec City, Canada

Welcome to the Proceedings of the 2012 Constraint Programming Doctoral Programme, held in conjunction with the 18th International Conference on Principles and Practice of Constraint Programming (CP'12) in Québec City, Canada. The doctoral programme is open to (mostly doctoral) students in all areas related to constraint programming. All participants present work either within the doctoral programme or in the main technical programme of the conference.

The papers in this proceedings are those which have been submitted directly to the doctoral programme. They contain a wide variety of work, either completed or in progress, being undertaken by the current generation of PhD students. We also list all students who have papers accepted into the main conference.

The Doctoral Programme includes an invited talk on "How to be a leader in your research area" by Carla Gomes (Cornell) and a tutorial on "Optimization under uncertainty" by Warren Powell (Princeton).

The Doctoral Program would not be possible without the support of many people and organisations. In particular, we would like to thank the program committee members and the sponsors of the CP conference.

We hope you have a great time in Québec City!

Michele Lombardi and Standa Živný
Doctoral Programme Chairs, 2012

Programme Committee:

Students with papers in the main CP programme:

Vincent Armant
Ignacio Castiñeiras
Kathryn Francis
Arnaud Letort
Jingying Li
Roberto Castañeda Lozano
Jean-Baptiste Mairy
Terrence Mak
Thiago Serra
Mohamed Siala
Robert Woodward
Wei Xia
Zhu Zhu

Table of Contents

Production Scheduling using Constraint Programming

Burcu Caglar Gencosman*, Cenk Ozmutlu*, Brahim Hnich**

*Uludag University, Turkey, **Izmir Economy University, Turkey

Abstract. In recent years, researchers have become increasingly interested in constraint programming (CP) applications in scheduling problems. However as the number of theoretical studies have increased the concept of practical usage of CP has not been investigated sufficiently. In this paper, we consider a scheduling problem at a mold company. We defined the problem using CP. Computational results show that CP generates successful schedules in minutes; hereby we conclude that CP can be used to obtain optimal solutions for real size scheduling problems.

Keywords: Production scheduling, constraint programming, mold production and metal forming.

1 Introduction

To date, there has been wide interest in scheduling problems of real production systems. Determining an approach for real-life scheduling problems has always been an interesting research area for many researchers. However, under the variability of real system and its dynamic structure, it becomes challenging to make efficient scheduling due to time constraints. There is now much evidence to support the hypothesis that finding optimal solutions of scheduling problems in acceptable times by mixed integer programming (MIP) has been challenging because of its complexity and stochastic nature [8], therefore applied researchers have become increasingly interested in generating feasible schedules in minutes by heuristic methodologies.

According to the literature, thousands of studies about different kinds of scheduling problems such as job shop, flow shop, open shop, and parallel machine scheduling have been studied by different researchers since 1950s [1]. In recent years, applied researchers have been studying about heuristics and metaheuristic to generate successful solutions of scheduling problems in reasonable times instead of finding optimal solutions. Gutierrez and Garcia-Magarino [2] developed a hybrid genetic algorithm for a flexible job-shop scheduling problem. Lei [3] built a particle swarm optimization methodology for multi-objective job shop scheduling problems with fuzzy processing time and due date. These meth-

odologies can be extended with tabu search, simulated annealing, and some combinations of two heuristic methods [4].

On the other hand, CP has become an important tool of generating exact solutions of scheduling problems [5], and there has been enhancement in CP applications of scheduling problems [6, 7]. Khayat et al. [8] developed a mixed integer mathematical programming model and a CP model to solve their combined production and material handling scheduling problems. The researchers also tested the CP limitations using 10x10 Fisher & Thompson scheduling problem which have not been able to solve more than 20 years. The optimal solution is found in 30 minutes with CP, and the optimality is approved in 2 hours.

The literature shows that optimal schedules could not be obtained by MIP, and researchers have been tending to develop heuristic methods in order to find feasible schedules [8]. Recently there has been a spate of interest in how to apply CP effectively to scheduling problems. The aim of this study is to demonstrate that CP can be an efficient way to obtain successful solutions for real size scheduling problems in acceptable times (less than an hour). We first define the problem and generate some examples according to real production environment. Then we develop a CP model in order to find optimal solutions of the examples. Finally we gather our findings and show that CP can be able to use in real size scheduling problems efficiently.

2 Scheduling Problem

In this study a real scheduling problem at a mold production and metal forming company is considered. The company produces automotive parts in a press line for main automotive companies. In current system, various jobs which have different number of operations are being produced at presses which are settled as a production line, and every operation has to be assigned to a different press, since each operation requires a specific mold to be attached to the press[1]. A number of O_i operations has to be performed to complete job *i*. Operation *j* of job *i*, can be processed by any machine, for a given processing time, however all of the operations of job *i* have to be assigned to adjacent machines. It is possible to produce multiple jobs simultaneously as long as total number of operations is less than the total number of machines. For example, three unique products, which have 5, 5, and 3 operations in sequence, can be produced in a line consists of 13 machines such as shown in Figure 1.

M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13
A.1	A.2	A.3	A.4	A.5	B.1	B.2	B.3	B.4	B.5	C.1	C.2	C.3

Figure 1: Loading 3 different products to the machines

[1] We will use “machine” instead of “press” in order to compatible with the scheduling literature.

Although the problem seems like a flow shop scheduling, before loading the molds every machine is identical, so the production line consists of parallel machines. After loading the molds or in other words, assigning the jobs to the machines it becomes separate flow shop problems. Therefore we build this unique problem according to its own properties. The following assumptions are made for the problem: The processing times are known and fixed. Every operation can be made in every machine. The decision variable *round* is related to setup and when a setup performs, the number of *round* increases. Molds can be changed when a production is completed or it can be delayed until all the productions in a *round* are completed reflecting a cost to objective function. Preemption of jobs is not allowed.

System constraints can be summarized as: Operations of jobs should be performed in adjacent machines. A machine can process only one operation at a time. An operation can be produced by only one machine at a time period. Demands have to be met. Enduring a penalty cost, products can be produced more than their demands. Sequence of operations has to ensure the precedence relationships between operations of a job. In order to find optimal schedules, we will build its CP model in following section.

3 Solving the scheduling problem using CP

CP guarantees the optimality if exist when giving sufficient times. CP reduces feasible solution space according to its constraints and domains of variables, and it finds optimal solution after trying all possible solutions. Due to different assessment methodology, we developed our CP model in a different perspective. For example we define *round* for changes in a period. A new *round* is created when at least one setup occurs where all machines are available, and production of some products can be continued or new products can be assigned to the machines in the *round.* Before declaring the constraints we first describe the notation of the model in following.

Indices and sets:

i: is a job from the set of all jobs ϵ *N={1,…n}*where n is the number of jobs.

r: number of rounds which is bounded by number of jobs ϵ *N={1,…|N|}*

j: number of operations ϵ *J={1,…|J_n|}*

m: is a machine from the set of all machines ϵ *M={1,…|M|}*

Parameters:

We produce a set for defined X decision variables' indices: Set: {<i,j,r> | i in N, j in O_i, r in N }.

O_i: Number of operations of job *i*.

R_{ij}: Processing time of operation *j* which is belonging to job *i*.

D_i: Demand of job *i*.

C_i: Completion time of job *i* that can be calculated as:$C_i = \sum_j R_{ij} + (D_i - 1) * R_{i,j_{max}}$

H_K: Cost of setup for each job.

H_W: Cost of waste.
H_R: Cost of round.
Decision Variables:

$$p_{ir} = \begin{cases} 1, if\ job\ i\ is\ performed\ at\ round\ r \\ 0, otherwise \end{cases}$$

x_{ijr}: Sequence of operation j of job i at round r. $x_{ijr} \in Set$
k_i: Amount of performed setups for job i.
w_i : Amount of waste for job i.
t_r: Total duration of round r.
v: Completion time of schedule.
The CP model is defined as follows:

$$\text{Min } z = v + \sum_i k_i * H_K + \sum_i w_i * H_W + \sum_r \sum_i p_{ir} * r * H_R \quad (1)$$

$$w_i = \sum_r p_{ir} * t_r - C_i\ , \forall\ i\ \in N \quad (2)$$

$$p_{ir} > 0 \xrightarrow{yields} t_r \leq \max(max_i(C_i * p_{ir}), 0)\ , \forall\ i, r\ \in N \quad (3)$$

$$\sum_r p_{ir} * t_r \geq C_i,\ \forall\ i\ \in N \quad (4)$$

$$t_r \geq t_{r+1},\ \forall\ r\ \in N, |r < N| \quad (5)$$

$$count(x_{ijr}, m) \leq 1, \forall\ r \in N, \forall\ m \in M \quad (6)$$

$$p_{ir} = 0 \xrightarrow{yields} x_{ijr} = 0,\ \forall\ i, j, r\ \in Set \quad (7)$$

$$p_{ir} = 1 \xrightarrow{yields} x_{ijr} \neq 0,\ \forall\ i, j, r\ \in Set \quad (8)$$

$$p_{ir} = 1 \xrightarrow{yields} x_{ij-1r} = x_{ijr} - 1, \forall\ i, j, r\ \in Set, j - 1 > 0 \quad (9)$$

$$(p_{ir-1} = 1\ AND\ p_{ir} = 1\ ) \xrightarrow{yields} x_{ijr-1} = x_{ijr}, \forall\ i, j, r\ \in Set, r - 1 > 0 \quad (10)$$

$$(p_{ir} = 1\ AND\ p_{iz} = 1\ ) \xrightarrow{yields} (p_{iz} = 0\ AND\ p_{ir+1} = 1\ ), \forall\ i, j, r\ \in Set, z\ \in N, z - r > 1 \quad (11)$$

$$(p_{ir} = 1\ \&\ p_{ir+1} = 1) \xrightarrow{yields} k_i = \sum_{r, r<n} O_i * \left(x_{ijr} \neq x_{ijr+1}\right),\ \forall\ i, j, r\ \in Set, r < n \quad (12)$$

$$\sum_i p_{ir} * O_i \leq M,\ \forall\ r \in N \quad (13)$$

$$\sum_r t_r = v \quad (14)$$

The objective function contains costs of setup, waste and *round* in addition to maximum completion time of the schedule. The cost of idle machines is represented by w_i decision variable, and we want to load all machines at a *round* in order to assure maximum workload. In addition, we want to load *rounds* as much as we can, so with the $\sum_r \sum_i p_{ir} * r * H_R$ expression, the *round* cost will increase when the index of *round* increase. Thus objective function forces the model to load jobs to smaller *rounds* if available. According to constraint (2), w_i occurs when duration of a *round* is greater than completion time of job *i*. Constraint (3) claims that if a job is performed at *round r*, the duration of this *round* should be less than or equal to its completion time. In addition, if job *i* is performed at *round r*, the duration of this *round* should be greater than or equal to its completion time which is represented by constraint (4). Constraint (5) ensures maximum workload for the previous *rounds*, and also breaks the symmetry to reduce the size of the problem. One of the important assumptions in this problem is that every machine at a round can process only one operation at the same time which is expressed by constraint (6). Constraints (7, 8) guarantee that if a job is not produced at *round r*, its sequence at this *round* should be zero. It is noticed that these two expressions assure the same issue, but in order to accelerate the CP

algorithm and reduce the branches number -enduring increased constraint number-, we choose to write constraint (8) too. Constraint (9) indicates that operations of a job should be assigned to adjacent machines. Constraint (10) guarantees that if a job is produced at sequential *rounds*, its position stays same, and constraint (11) prevents production of the same product in non-sequential *rounds*. Constraint (12) calculates total number of setups for each job. Constraint (13) represents that total operations assigned to a *round* should be less than or equal to number of machines, and finally constraint (14) calculates completion time of all production horizon.

CP searches the solution space according to its search procedure, and assigns values to the variables regarding that procedure. There is a strong relationship between success of a CP and its search strategy; however performance of the strategies can be different through the problem types. For example a search strategy can be performed well for a vehicle routing problem, but it can be failed when solving a scheduling problem. OPL language provides various search procedures for the users, and the default search strategy is called "first fail principle". We ran our model with this strategy and summarized the results in Table 1.

In order to test the CP model, we developed 12 different instances (can be shared with interested researchers). P1-P8 instances generated using uniform data and G1-G4 is generated using real production data. Tests are performed on a PC with 2.13 GHz processor and 6 GB RAM. The CP model is built using ILOG CPLEX 12.2. It is important to mention that we used the total completion time of production horizon which can be used for comparisons with the literature. In order to gather results of all problems, CP model is limited with 1800 seconds for big size problems. Experimental results are summarized in Table 1. As shown in Table 1, last column gives the values of decision variables in optimal solution or in an upper bound that represents the total completion time of production horizon.

Table 1: Experimental results of MIP and CP models

Prob.	**Jobs**	**# of Operations**	**# of Machines**	**CPU time(sec)**	$\sum_r t_r$
P1	3	5-(2,2,1)	5	0.02	2002
P2	4	10-(2,3,3,2)	5	0.02	4008
P3	4	10-(2,3,3,2)	10	0.02	2004
P4	5	15-(2,3,3,4,3)	10	0.02	4008
P5	5	15-(2,3,3,4,3)	15	0.03	2006
P6	6	20-(3,3,5,4,3,2)	15	0.05	4008
P7	6	20-(3,3,5,4,3,2)	20	0.03	2008
P8	7	25-(3,3,5,4,3,2,5)	20	0.08	4008
G1	5	12-(1,1,4,4,2)	13	0.16	43254
G2	10	24-(1,1,4,4,2,2,1,1,4,4)	13	0.20	**86508**
G3	15	29-(1,1,4,4,2,2,1,1,4,4,1,1,1,1,1)	13	1800	**115266**[1]
G4	20	43-(1,1,4,4,2,2,1,1,4,4,1,1,1,1,1,3,3,3,3,2)	13	1800	**134508**[2]

[1]: It is found after 5.11 seconds.[2]: It is found after 16.8 seconds.

Considering CPU times, for P1-G2 problems CP reaches optimal solutions within seconds. Even though CP could not prove optimal solution for G3 and G4 problems, it finds feasible solutions within seconds, which might be an optimal solution.

4 Conclusion

In this study, it is demonstrated that we can find feasible solutions with CP. The present study has raised a number of interesting findings, but a larger research is needed to establish how successful CP is. Future research could be focused on comparing CP model with other exact approaches like mixed integer programming. Further, the proposed method can be extended to more general scheduling problems.

5 References

[1] Allahverdi, A., Ng, C.T., Cheng, T.C.E., Kovalyov, M.Y. : A survey of scheduling problems with setup times or costs. European Journal of Operational Research, 187: 985-1032 (2008).

[2] Gutierrez, C., Garcia-Magarino, I.: Modular design of a hybrid genetic algorithm for a flexible job-shop scheduling problem. Knowledge-Based Systems, 24 (1): 102-112 (2011).

[3] Lei, D.: Pareto archive particle swarm optimization for multi-objective fuzzy job shop scheduling problems. International Journal of Advanced Manufacturing Technology, 37 (1-2): 157-165 (2008).

[4] Zhang, C. Y., Li, P., Rao, Y., Guan, Z.: A very fast TS/SA algorithm for the job shop scheduling problem. Computers\&Operations Research 35 : 282-294 (2008).

[5] Van Hentenryck, P.: The OPL optimization programming language. Cambridge, Massachusetts: The MIT Press (1999).

[6] Baptiste, P., Le Pape, C., Nuijten, W. : Constrained-based scheduling: Applying constraint programming to scheduling problems. New York: Springer (2005).

[7] Nuitjen, W. P. M., Aarts, E. H. L.: A computational study of constraint satisfaction for multiple capacitated job shop scheduling. European Journal of Operational Research, 90, 269-284 (1996).

[8] Khayat G., Langevin A., Riopel D.: Integrated production and material handling scheduling using mathematical programming and constraint programming. European Journal of Operational Research, 175: 1818-1832 (2006).

[9] Zeballos, L. J., Novas, J. M., Henning, G. P.: A CP Formulation for Scheduling Multiproduct Multistage Batch Plants. Computers and Chemical Engineering, 35: 2973-2989 (2011).

Energy optimization of metro timetables: a hybrid approach

David Fournier[1] [2], Denis Mulard[1], and François Fages[2]

[1] General Electric Transportation ICS Delta
Tour Europlaza 28D5, 20 avenue André Prothin 92063 Paris La Défense Cedex (France) denis.mulard@ge.com
[2] INRIA Rocquencourt Equipe Contraintes
Domaine de Voluceau Bâtiment 8 Rocquencourt BP 105 78153 Le Chesnay (France)
{david.fournier,francois.fages}@inria.fr

Abstract. In a metro line, metros are capable of generating electricity by braking. This energy, immediately available in the third rail, is lost if no metro in the neighbourhood can consume it. Thus it is possible to decrease total energy consumption by synchronizing accelerations and braking.
We describe a model for optimizing energy consumption which does not alter significantly quality of service by subtly modifying dwell times.
A hybrid genetic/linear programming algorithm has been implemented to tackle this problem and compute the distribution of braking metros.
On a typical example, the savings are more than 6%.

1 Introduction

Sustainable energy has been a major issue over the last years. Transportation is a major field concerned about energy consumption and companies in this industry try optimize as much as possible the energy consumption of their solutions, in particular in mass rapid transit such as metros. Several hardware solutions, like fly-wheels or super capacities have been developed to save energy. However, these solutions involve buying and maintaining potentially costly material which can be difficult to justify economically.

Besides that, software solutions have been implemented, controlling finely different parameters to minimize energy consumption. However, optimizing a timetable is known as highly combinatorial and some formulations have been proven to be NP-complete [2]. Consequently, it is only possible to search for local optima as soon as instances get large. Albrecht implements in [1] a genetic algorithm and shows it is possible to reduce energy consumption and more particularly power peaks using reserve time of metros. However this optimization is done modifying metro speed patterns, which can be complicated to implement in real-time. Kim et al. [4] propose a MILP model to optimize metro time departures. This model is interesting but it is too much simplified - even if it is necessary due to the complexity of the problem - to correspond to reality.

Eventually, Nasri et al. [5] show that it may be possible to decrease energy consumption by modifying dwell times.

We describe in this article a method which modifies dwell times to synchronize accelerations and braking of metros. Dwell times have the advantage to be easy to update in timetabling software. To do that, we use a genetic algorithm to minimize an objective function - corresponding to the global energy consumption over a time horizon - computed with a linear program.

2 Problem description

The energy consumption in a metro line can be decreased by synchronizing braking and accelerations of metros. Indeed, an electric motor behaves as a generator when braking by transforming its kinetic energy into electrical energy. This energy, available in the third rail, has to be absorbed immediately by another metro in the neighbourhood. If not, it is dissipated as heat and then is lost.

Anyway, the distance between metros which are generating energy and candidate metros induces that part of the transferred regenerative energy is lost in the third rail due to Joule's effect.

Most timetables don't take into account energy issues. They usually have been created to maximize quality of service, complying with security and other constraints like drivers' shift or weekend periods for instance. That's why it is hard to create a brand new timetable only to take into account energy issues. It is however possible to slightly modify current timetables to include some energy optimization.

What we try here is to minimize energy consumption of a metro line during a given time horizon by modifying the off-line timetable.

3 Data

The model is restricted to a single metro line (no fork or loops) including 31 stations with two terminals A and B. All trips are done from A to B or B to A, stopping at all stations.

The timetable, based on real data, is a bit more detailed than the one given to passengers; in addition to departure times at every station, it compiles also:

- running times between every station.
- dwell times at every station.

Dwell times represent the nominal waiting time of a metro in a given station. We consider here that every metro have the same dwell time for a given station, not depending on the hour of the day.

For every timeslot (1 second in our model), we know the position of metros (between which stations they are) and the energy they consume (arbitrary positive) or produce (negative energy). Contrary to timetables data which are real, energy data have been created following energy models like the one depicted in

[4]. Units are arbitrary: a value of 1 in this system corresponds to the energy consumed by a metro at full throttle during one second.

Losses due to Joule's effect are compiled in an efficiency matrix. It details the percentage of energy which can be transferred from a point to another point in the line.

4 Model

4.1 Timetable

The objective (1) is to minimize the energy consumption over a given time period, thus to minimize the sum of energy consumptions over every timeslot. If we consider T the set of timeslots and y_t the energy consumption of the line at timeslot t, then our objective function is:

$$\min \sum_{t \in T} y_t \tag{1}$$

The computation of y_t can be seen as a formulation of a generalized max-flow problem in a lossy network [6] which can be formulated as an LP problem.

As we optimize global energy consumption by modifying dwell times, we need to clarify what are the relevant dwell times for our formulation. We compute them as follows:

Sets

- T: timeslots.
- I: metros.
- S: stations.
- $\mathcal{D}^r \subset I \times S$: relevant dwell times.

Parameters

- $Dep_{i,s}$: arrival time $t \in T$ of $i \in I$ to the station $s \in S$.
- $D_{i,s}$: dwell time of $i, s \in \mathcal{D}^r$.
- δ: minimal quantity for decreasing/increasing a dwell time.

Variables

- $d_{i,s}$: optimized dwell time of metro $i \in I$ at station $s \in S$.
- $n_{i,s} \in \mathbb{Z}$: number of times δ is applied to a dwell time i, s.

Model

$$d_{i,s} = D_{i,s} + n.\delta \tag{2}$$

with
$$\mathcal{D}^r = \{D_{i,s} \in I \times S / \inf(T) \leq Dep_{i,s} \leq \sup(T)\} \tag{3}$$

Then these are the dwell times $d_{i,s} \in \mathcal{D}^r \subset I \times S$ that the genetic algorithm will modify in order to minimize the objective function. Note that n can be unbounded. In our model, it is however bounded by small integers to stick on the quality of service issue and to keep having an *invisible* optimization for the final user.

4.2 The instantaneous objective function

Modifying dwell times involves a new synchronization between metros. We thus have to compute at every iteration of the genetic algorithm, the resulting objective function. As explicated in (1), every timeslot represents an independent problem. The issue here is that it is hard to know exactly how regenerated energy will spread throughout third rail and other metros. [3] takes as an hypothesis that metros can transfer entirely their regenerative energy to others only if they belong to the same electric sub-station.

We are making the hypothesis that energy is dissipating proportionally to the distance between two metros. We are also taking as hypothesis that the energy is spread in an optimal way, i.e. the model minimizes the loss of energy.

Then, for a given timeslot we have :

Sets

- I^+: metros consuming energy.
- I^-: metros producing energy.

Parameters

- E_i^+: energy consumed by metro $i \in I^+ (> 0)$.
- E_i^-: energy produced by metro $i \in I^- (< 0)$.
- $A_{i,j}$: proportion of the energy produced by $i \in I^-$ transferable to $j \in I^+$ due to Joule's effect.

Variables

- $x_{i,j}$: proportion of the energy produced by $i \in I^-$ transferred to $j \in I^+$.

Model

$$\text{minimize} \qquad y \tag{4}$$

subject to

$$\sum_{i}^{I^+} E_i^+ + \sum_{i}^{I^-} (E_i^- . \sum_{j}^{I^+} x_{i,j} . A_{i,j}) \leq y \tag{5}$$

$$\sum_{j}^{I^+} x_{i,j} \leq 1 \quad \forall i \in I^- \tag{6}$$

$$-\sum_{i}^{I^-} x_{i,j} . E_i^- . A_{i,j} \leq E_j^+ \quad \forall j \in I^+ \tag{7}$$

$$x_{i,j} \geq 0 \quad \forall i \in I^-, \forall j \in I^+ \tag{8}$$

$$y \geq 0 \tag{9}$$

The LP model minimizes the energy consumed by spreading the energy produced in such a way $-\sum_i^{I^-}(E_i^-.\sum_j^{I^+} x_{i,j}.A_{i,j})$ is maximized. Note that (9) prevents the energy to be less than 0 at a given timeslot. It is because we consider that the regenerative energy which is not utilized immediately is lost.

5 Genetic algorithm

By modifying only slightly the dwell times, we consider that the algorithm never reaches non satisfiability as we stay in tolerable intervals, e.g. for headways.

Every individual in the population is represented by a two-arrays table with metros in rows and stations in columns. Each cell represents a dwell time.

Starting with initial dwell times, we create a population made of 100 individuals. Then every dwell time is randomized within a predefined domain, e.g. {-3s, 0s, +3s, +6s, +9s}. Finally every iteration, individuals are classified according to their objective function and selected. We apply crossover and mutation to them until convergence.

6 Results

6.1 Computation time

Our model has been tested with a one-hour time horizon, corresponding to 3600 timeslots, 29 metros and 495 dwell times to optimize. The objective function has a value 8504 a.u. (arbitrary unit) at time t_0. After 450 iterations, total energy consumption is only 7939,4 a.u, that to say 6,6% saving.

The computation lasts over 88 hours long on a Intel Core 2 1.86GHz Linux PC. As this optimization is to minimize an off-line timetable, we can allow it.

6.2 Robustness

A real metro line is subject to minor perturbations that can affect the adherence to the timetable.

To check the relevance of the optimization, we have added a random noise on optimized dwell times to quantify the robustness of the objective function. This noise consists in randomly modifying dwell times by $\pm\delta s$.

Noise (s)	1	3	6
Average on 100 tries (a.u.)	7964,9	7995,7	8028,4
Saving (%)	6,3	6,0	5,6

Table 1. Alteration of the objective function according to noise

Table 1 shows the results. We can see that even with 6 second noise, The optimization is still saving 5,6% energy. This means that the optimized solution is saving energy, but also all its neighbour solutions.

7 Perspectives

This resolution method to optimize the energy consumption in a metro line seems promising and deserves more research. In particular, we want to increase the number of parameters we could modify, such as departure times in terminals or speed profiles.

We will make effort also to compare these results with other methods such as MILP or constraint programming.

Eventually, decreasing computation time will allow this method to be used in a real-time context, in particular when it is about to optimize energy consumption after major incidents.

References

1. T. Albrecht. Reducing power peaks and energy consumption in rail transit systems by simultaneous metro running time control. *Computers in Railways IX*, 2004.
2. E. Bampas, G. Kaouri, M. Lampis, and A. Pagourtzis. Periodic metro scheduling. *ATMOS 2006*, 2006.
3. K.M. Kim, K.T. Kim, and M.S. Han. A model and approaches for synchronized energy saving in timetabling. *WCRR 2011*, May 2011.
4. Kyung min Kim, Suk mun Oh, and Moonseob Han. A mathematical approach for reducing the maximum traction energy: The case of korean mrt trains. *IMECS 2010*, March 2010.
5. A. Nasri, M. Fekri Moghadam, and H. Mokhtari. Timetable optimization for maximum usage of regenerative energy of braking in electrical railway systems. *SPEEDAM 2010*, 2010.
6. M. Shigeno. A survey of combinatorial maximum flow algorithms on a network with gains. *Journal of the Operations Research Society of Japan.*

Generalized Micro-Structures for Non-Binary CSP*

Achref El Mouelhi

LSIS - UMR CNRS 7296
Aix-Marseille Université
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
achref.elmouelhi@lsis.org

Abstract. In [6], a new approach has been introduced for improving the solving of binary CSPs. This approach is based on the notion of micro-structure for binary CSPs. This formalism was used to analyze the theoretical complexity of binary CSPs.
In this paper, we generalize the notion of micro-structure to non-binary CSPs in order to retain the tractability as in the binary case.

1 Preliminaries

Constraint Satisfaction Problems (CSPs [10]) provide an efficient way of formulating problems in computer science, especially in Artificial Intelligence such as scheduling, temporal reasoning, planning, graph problems, just to name a few.

Formally, a *constraint satisfaction problem* is a triple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of variables, $D = (D_{x_1}, \ldots, D_{x_n})$ is a list of finite domains of values, one per variable, and $C = \{C_1, \ldots, C_e\}$ is a finite set of constraints. Each constraint C_i is a couple $(S(C_i), R(C_i))$, where $S(C_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ is the *scope* of c_i, and $R(C_i) \subseteq D_{x_{i_1}} \times \cdots \times D_{x_{i_k}}$ is its *relation*. The *arity* of C_i is $|S(C_i)|$. A CSP is called *binary* if all constraints are of arity 2 (we denote C_{ij} the binary constraint whose scope is $S(C_{ij}) = \{x_i, x_j\}$). Otherwise, a CSP is said to be *n-ary*. A *consistent assignment* is an assignment that does not violate the constraints. A *solution* is a complete assignment that satisfies all constraints. Testing whether a CSP has a solution is known to be NP-complete.

We can associate to a binary CSP a graph called a micro-structure which is defined as follows:

Definition 1 (micro-structure [6]) *Given a binary CSP $P = (X, D, C)$, the* micro-structure *of P is the undirected graph $\mu(P) = (V, E)$ with:*

- $V = \{(x_i, v_i) : x_i \in X, v_i \in D_{x_i}\}$,
- $E = \{\ \{(x_i, v_i), (x_j, v_j)\}\ \mid\ i \neq j, C_{ij} \notin C$ *or* $C_{ij} \in C, (v_i, v_j) \in R(C_{ij})\}$

The micro-structure was introduced by Jégou in order to detect new tractable classes for CSP based on graph theory. In [2], Cohen showed that the class of binary CSPs with triangulated complement of micro-structure is tractable and arc-consistency is a decision procedure.

* This work was supported by the French National Research Agency under grant TUPLES (ANR-2010-BLAN-0210).

After that, Jégou and al. in [8] presented new results on the effectiveness of classical algorithms when the number of maximal cliques in graph of micro-structure can be bounded by a polynomial.

Therefore, the goal of this report is to generalize the definition of micro-structure to non-binary CSPs.

2 Generalized Micro-Structures

The first extension of the notion of micro-structure to non-binary CSPs was proposed by Cohen in [2]: this generalization is based on hypergraphs [3]. In contrast, our generalisations are inspired by technical methods of conversion between non-binary and binary CSPs [12, 1]. Then, our micro-structure are a simple undirected graph obtained by using binary encoding based on graphic representation of non-binary CSPs: the well known methods are the dual encoding (also called dual representation), hidden transformation (also called hidden variable representation) and mixed encoding.

2.1 Generalized micro-structure based on dual representation

The dual encoding was introduced by Dechter and Pearl in [4] and is based on the dual graph representation (also called intergraph in [5, 7] or line graph in graph theory) which comes from the relational database. In this encoding, the constraints of the original problem become variables (also called dual variables). The domain of each new variable is exactly the set of tuples allowed by the original constraint. We define a constraint between two dual variables if the original constraints share at least one variable.

Definition 2 (generalized micro-structure based on dual representation) *Given a CSP $P = (X, D, C)$ (not necessarily binary), the* generalized micro-structure *of P is the undirected graph $\mu_{G_d}(P) = (V, E)$ with:*

- $V = \{(C_i, t_i) : C_i \in C, t_i \in R(C_i)\}$,
- $E = \{\ \{(C_i, t_i), (C_j, t_j)\}\ \mid\ i \neq j, t_i[S(C_i) \cap S(C_j)] = t_j[S(C_i) \cap S(C_j)]\}$

As for the micro-structure, there is a direct relationship between cliques and solutions of CSPs:

Theorem 1 *A CSP P has a solution iff $\mu_{G_d}(P)$ has a clique of size e.*

Proof: By construction, $\mu_{G_d}(P)$ is e-partite, and any clique contains at most one vertex (C_i, t_i) per constraint $C_i \in C$. Hence the e-cliques of $\mu_{G_d}(P)$ correspond exactly to its cliques with one vertex (C_i, t_i) per constraint $C_i \in C$. Now by construction of $\mu_{G_d}(P)$ again, any two vertices (C_i, t_i), (C_j, t_j) joined by an edge (in particular, in some clique) satisfy $t_i[S(C_i) \cap S(C_j)] = t_j[S(C_i) \cap S(C_j)]$. Hence all t_i's in a clique join together, and it follows that the e-cliques of $\mu_{G_d}(P)$ correspond exactly to tuples t which are joins of one allowed tuple per constraint, that is, to solutions of P. □

The example below will be used throughout the paper:

Example 1 *The figure 1 presents a CSP $P = (X, D, C)$ with :*
$X = \{x_1, x_2, x_3, x_4, x_5\}$,
$D = (D_{x_1}, D_{x_2}, D_{x_3}, D_{x_4}, D_{x_5})$ *with*

$D_{x_1} = \{a, a'\}$, $D_{x_2} = \{b\}$, $D_{x_3} = \{c\}$, $D_{x_4} = \{d, d'\}$ *and* $D_{x_5} = \{e\}$.
$C = \{C_1, C_2, C_3, C_4\}$ *is a set of four constraints with* $S(C_1) = \{x_1, x_2\}$, $S(C_2) = \{x_2, x_3, x_5\}$, $S(C_3) = \{x_3, x_4, x_5\}$ *and* $S(C_4) = \{x_2, x_5\}$.
The relations associated to the previous constraints are given by these tables:

$R(C_1)$	
x_1	x_2
a	b
a'	b

$R(C_2)$		
x_2	x_3	x_5
b	c	e

$R(C_3)$		
x_3	x_4	x_5
c	d	e
c	d'	e

$R(C_4)$	
x_2	x_5
b	e

The generalised micro-structure based on dual encoding of the last example is shown in figure 1.

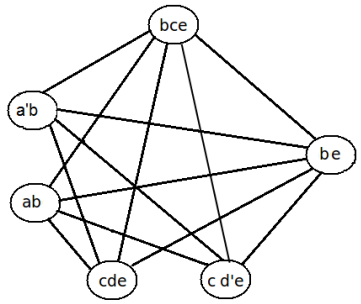


Fig. 1. Generalized micro-structure built by using dual representation.

We have four constraints, then $e = 4$. Thanks to Theorem 1, a solution of P is a clique of size 4, e.g. (ab, bce, be, cde).

The generalized micro-structure based on minimal intergraph [5, 7] is similar to generalized micro-structure based on dual encoding because the deleted edges in the minimal intergraph will be added by the relation $t_i[S(C_i) \cap S(C_j)] = t_j[S(C_i) \cap S(C_j)]$.

2.2 Generalized micro-structure based on hidden transformation

The hidden variable encoding was inspired by Peirce [9] (cited in [11]). In hidden transformation, the set of variables contains the original variables plus the set of dual variables. There is a binary constraint between a dual variable and original variable if the original variable belongs to the scope of dual variable [12].

Definition 3 (generalized micro-structure based on hidden transformation) *Given a CSP* $P = (X, D, C)$, *the* generalized micro-structure based on hidden transformation *of* P *is the undirected graph* $\mu_{G_h}(P) = (V, E)$ *with:*

- $V = S_1 \cup S_2$
- $S_1 = \{(C_i, t_i) : C_i \in C, t_i \in R(C_i)\}$,
- $S_2 = \{(x_j, v_j) : x_j \in X, v_j \in D_{x_j}\}$,
- $E = \{\ \{(C_i, t_i), (x_j, v_j)\}\ \mid$ *either* $x_j \in S(C_i)$ *and* $v_j = t_i[x_j]$ *or* $x_j \notin S(C_i)\}$

We now turn to relationship between bicliques and solutions of CSPs. Before that, we should recall that a biclique is a complete bipartite graph, i.e. a bipartite graph in which every vertex of the first set is connected to all vertices of the second set.

Proposition 1 *In a generalized micro-structure, a $K_{n,e}$ biclique with e tuples, such that no two tuples belong to the same constraint, cannot contain two different values of the same variable.*

Proof: We assume that a $K_{n,e}$ biclique with e tuples, such that no two tuples belong to the same constraint, can contain two different values v_j and v'_j of the same variable x_j. Therefore, there is at least a constraint C_i such that $x_j \in S(C_i)$. Thus, $t_i[x_j] = v_j$, v'_j or another $v"_j$ and in all three cases, the assumption is false because t_i cannot be connected to two different values of the same variable. □

Proposition 2 *In a generalized micro-structure, a $K_{n,e}$ biclique with n values, such that no two values belong to the same domain, cannot contain two different tuples of the same constraint.*

Proof: We assume that a $K_{n,e}$ biclique with n values, such that no two values belong to the same variable, can contain two different tuples t_i and t'_i of the same constraint C_i. Therefore, there is at least a variable x_j such that $t_i[x_j] \neq t'_i[x_j]$. If $t_i[x_j] = v_j$ and $t'_i[x_j] = v'_j$ must belong to the same biclique. Thus, the assumption is false because we cannot have two values of a same variable.□

Using these two properties, we can deduce the following theorem :

Theorem 2 *Given a CSP $P = (X, D, C)$ and $\mu_{G_h}(P)$ its generalized micro-structure, P has a solution iff $\mu_{G_h}(P)$ has a $K_{n,e}$ biclique with n values and e tuples such that no two values belong to the same domain and no two tuples belong to the same constraint.*

Proof: A $K_{n,e}$ biclique with n values and e tuples such that no two values belong to the same domain and no two tuples belong to the same constraint correpond to a consistent assignment which satisfies all constraints. □

Figure 2 represents the generalized micro-structure based on hidden transformation of the CSP of example 1.

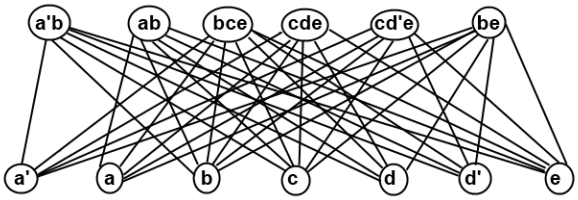


Fig. 2. Generalized micro-structure built by using hidden variable.

Based on the previous example, we can easily see that a biclique does not necessarily correspond to a solution. Although, $\{a, a', b, c, e, ab, ab', bce, be\}$ is $K_{5,4}$ biclique but it is not a solution. Contrary to $\{a, b, c, d, e, ab, bce, be, cde\}$ that is a $K_{5,4}$ biclique and is a solution of P.

Then, the set of solutions is not equivalent to the set of bicliques. This is due to the manner which the graph of micro-structure is completed. For the

next generalization, we will propose another manner to complete the graph of micro-structure: this new way of representation can also be deduced from hidden encoding.

2.3 Generalized micro-structure based on mixed encoding

We finally turn to the last interesting method of translation which is called mixed encoding. It allows us to connect the dual variables to the original variables, two tuples of two different constraints and two values of two different variables.

In other words, it uses at the same time dual encoding and hidden variable encoding. More description is given by the following definition.

Definition 4 (generalized micro-structure based on mixed encoding) *Given a CSP $P = (X, D, C)$, the* generalized micro-structure based on mixed encoding *of P is the undirected graph $\mu_{G_m}(P) = (V, E)$ with:*

- $V = S_1 \cup S_2$
- $S_1 = \{(C_i, t_i) : C_i \in C, t_i \in R(C_i)\}$,
- $S_2 = \{(x_j, v_j) : x_j \in X, v_j \in D_{x_j}\}$,
- $E = E_1 \cup E_2 \cup E_3$
- $E_1 = \{\ \{(C_i, t_i), (C_j, t_j)\}\ \mid\ i \neq j, t_i[S(C_i) \cap S(C_j)] = t_j[S(C_i) \cap S(C_j)]\}$
- $E_2 = \{\ \{(C_i, t_i), (x_j, v_j)\}\ \mid\ \textit{either } x_j \in S(C_i) \textit{ and } v_j = t_i[x_j] \textit{ or } x_j \notin S(C_i)\}$
- $E_3 = \{\ \{(x_i, v_i),\ (x_j, v_j)\} \mid\ i \neq j \textit{ and } x_i \neq x_j\}$

The generalized micro-structure based on mixed encoding of the CSP of example 1 is shown in figure 3.

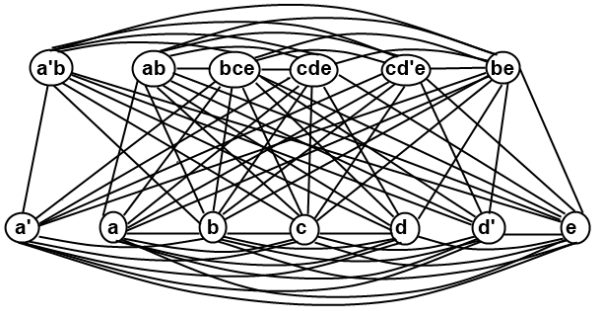


Fig. 3. Generalized micro-structure built by using mixed encoding.

Proposition 3 *In a generalized micro-structure, a K_{n+e} clique cannot contain more than n values where each pair of values derived from two different variables.*

Proof: We assume that proposition 3 is false, then a K_{n+e} clique must contain two values v_i and v_i' of the same variable x_j. Thus, the corresponding vertices to v_i and v_i' cannot be adjacent and cannot belong to the same clique. □

Proposition 4 *In a generalized micro-structure, a K_{n+e} clique cannot contain more than e tuples where each couple of tuples derived from two different constraint.*

Proof: (similar to the previous proof)

According to the last properties, there is a strong relationship between cliques and solutions of CSPs:

Theorem 3 *A CSP P has a solution iff $\mu_{G_m}(P)$ has a clique of size $n + e$.*

Proof: In a generalized micro-structure based on mixed encoding, a K_{n+e} clique correspond to a consistent assignment of n variables which satisfies e constraints (based on the two previous propositions). Then, it is a solution to P. □

The first micro-structure is exactly the micro-structure of the dual CSP, but the second and the last representations correspond neither to the micro-structure of hidden CSP nor to the micro-structure of a mixed CSP because of the way to complete these two graphs.

3 Conclusion

We have investigated the different binary representation of n-ary CSPs in order to define a generalization of micro-structure for non-binary CSPs. The first perspective is to check whether the results in [8] can be extended to non-binary CSPs represented by one of this micro-structure. The second aims to provide formal frameworks for studying properties on the complexity of n-ary CSPs as with the binary case.

References

1. Fahiem Bacchus, Xinguang Chen, Peter van Beek, and Toby Walsh. Binary vs. non-binary constraints. *AI*, pages 1–37, 2002.
2. David A. Cohen. A New Classs of Binary CSPs for which Arc-Constistency Is a Decision Procedure. In *CP*, pages 807–811, 2003.
3. David A. Cohen, Peter Jeavons, Christopher Jefferson, Karen E. Petrie, and Barbara M. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints*, 11(2-3):115–137, 2006.
4. R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.
5. P. Jégou. *Contribution à l'étude des problèmes de satisfaction de contraintes : Algorithmes de propagation et de résolution – Propagation de contraintes dans les réseau dynamiques.* PhD thesis, Université des Sciences et Techniques du Languedoc, January 1991.
6. P. Jégou. Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In *AAAI*, pages 731–736, 1993.
7. P. Jégou. On the Consistency of General Constraint-Satisfaction Problems. In *AAAI*, pages 114–119, 1993.
8. Achref El Mouelhi, Philippe Jégou, Cyril Terrioux, and Bruno Zanuttini. On the efficiency of backtracking algorithms for binary constraint satisfaction problems. In *ISAIM*, January 2012.
9. C.S. Peirce, C. Hartshorne, and P. Weiss. *Collected Papers of Charles Sanders Peirce.* Number vol. 3. Harvard University Press, 1933.
10. F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming.* Elsevier, 2006.
11. Francesca Rossi, Charles J. Petrie, and Vasant Dhar. On the equivalence of constraint satisfaction problems. In *ECAI*, pages 550–556, 1990.
12. K. Stergiou and T. Walsh. Encodings of Non-Binary Constraint Satisfaction Problems. In *AAAI*, pages 163–168, 1999.

CDF-Intervals: Reliable Constraint Reasoning with Quantifiable Information

Aya Saad[2], Carmen Gervet[1], and Thom Frühwirth[2]

[1] German University in Cairo,
[2] Universität Ulm Germany

Abstract. This paper defines a new domain for reasoning with uncertain data. It extends the usual interval arithmetic approach with a second dimension that captures the cummulative distribution function. Due to its monotonic property *cdf* forms by default a lattice that assists in a smooth domain reasoning. We further extend the *cdf*-interval approach to bound the probability distribution in a p-box notation. The idea is embodied as a new solver in the ECL^{i}PS^e system. Experimental results indicate that the solver brings solution insights compared to convex models, leading to a full enclosure of the data along with its probability distribution.

1 Introduction

Convex models are favored for their tractable computation. They are the heart of robust/reliable optimization [1] in Operation Research, mixed CSP [4], reliable constraint reasoning [9], and quantified CSP [10] in Constraint Programming. The majority of the techniques presented rely on interval reasoning to model problems with uncertain/ill-defined data and seek output solution sets that satisfy all possible realizations of data sought. Drawback of convex structures reveals in their solution set property which has an equal weight of knowledge. To overcome this problem, *cdf*-intervals were introduced in [6] to quantify knowledge. CDF has a monotonic nature; hence it defines a lattice in the second dimension: the degree of knowledge. The algebraic structure presented approximates the probability to the nearest uniform cummulative distribution. To guarantee the full encapsulation of the degree of knowledge, we further extend *cdf*-intervals by bounding the unknown probability. The new convex structure yields *cdf*-intervals in a p-box representation. Empirical results show better performance over the original work on a scale; furthermore, the model outputs an additional probabilitic information over reliable models.

This paper demonstrates the algebraic structures of *cdf*-intervals and the calculus behind. It is structured as follows: (1) the representation of uncertain data that is input to the *cdf*-intervals, (2) a framework for solving systems of arithmetic constraints over *cdf*-intervals, (3) an experimental evaluation and a show case for a system of linear constraints.

2 Interpretation of the Confidence Interval

CDF-Intervals [6] describe an unknown value along with its degree of knowledge in a linear format. Hence data whereabouts in the algebraic structure is approximated to the nearest uniform probability distribution. Cummulative distribution function *cdf* is elected due to its monotonic property that is suitable for interval computation in addition to its aggregated property that sufficiently propagates probability information to the bounds. The concept of p-boxes introduced in [8] when associated with *cdf*-intervals further bound the unknown probability distribution to ensure a full encapsulation of the provided knowledge.

The notion of p-box constitutes the heart of our new *cdf*-interval domain specification. A p-box is basically a convex structure over a set of probabilities.

Definition 1 (p-box). *Let* $\underline{F}$ *and* $\overline{F}$*, be two cumulative probability distributions.* $[\underline{F}_X, \overline{F}_X]$ *specifies the probability box of a random variable* X *whose distribution* F_X *is contained within the p-box bounds:*

$$\underline{F}_X(x) \leq F_X(x) \leq \overline{F}_X(x) \quad \forall x \in (-\infty, +\infty) \tag{1}$$

p-box bounds introduced in this work are choosen to be uniform in order to simplify the computation. We illustrate in Fig. 1 an observed collected data from a measuring instrument, its *cdf* that shapes a staircase function. We display algebraic representations: the *cdf*-intervals with linear/uniform probability approximation [6] and *cdf*-intervals augmented by a p-box with uniform bounds.

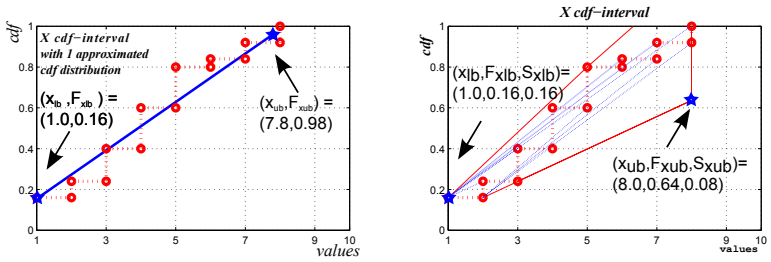


Fig. 1. *cdf*-interval: from a linear approximation to a p-box for X

Unlike real interval modeling *cdf*-intervals add up a quantitative information and define this information over a 2D space. CDF-Intervals store quantile bounds and their corresponding *cdf*-values; a point p_x is specified as $(x, F_x^p) \in \mathcal{U}$; due to its monotonic and aggregated properties we can extract the *cdf* approximated value that corresponds to any quantile lying within the interval bounds in an order of O(1). p-boxes define a triplet per bound. For a point p_x lying within the interval bounds specified as $(x, F_x^p, S_x^p) \in \mathcal{U}$, x is the quantile value, F_x^p its *cdf* value, and S_x^p the slope of the *cdf* distribution the point lies on. Thus quantitative

information for a quantile is defined by a probability range of values that lie within the two slopes bounding the p-box *cdf*-interval.

Fig. 1 illustrates the same variable X that ranges in the *cdf*-interval $[(1, 0.16), (7.8, 0.98)]$ and in the p-box $[(1, 0.16, 0.16), (8.0, 0.64, 0.08)]$. This indicates that the real interval representation of X is $[1.0, 8.0]$. The *cdf*-interval adds to this representation the quantitative information by approximating the distribution that ranges between [0.16,0.98]; p-box bounds the measured distribution by two slope triplets: quantile 1.0 has a chance of occurence for X that cannot exceed 16%, whilst X can be at most 8.0 with a probability that is at least 64%.

3 CDF-Intervals Solver

Constraint reasoning using the *cdf*-intervals behaves like a solver over real intervals, it is based on the relational arithmetic of real intervals where arithmetic expressions are interpreted as relations [2]. Relations over *cdf*-intervals and their p-box augmented version in turn are handled using transformation rules that extend those over real intervals with additional inferences in the *cdf*-dimension. We have implemented the main transformation rules for the basic interval relations $\{=_{\mathcal{U}}, \preccurlyeq_{\mathcal{U}}, +_{\mathcal{U}}, \times_{\mathcal{U}}\}$ over the p-box *cdf*-intervals. The solver is a separate module in the ECL^{i}PS^e [3] environment. A constraint system input to the solver fails if domain bounds do not preserve the ordering in the 2D space; real ordering is employed on interval quantiles and stochastic dominance ordering is used to order probabilities. The handling of the transformation rules is done by a relaxation algorithm which resembles the arc consistency algorithm AC-3 [5]. The solver converges to a fixed point or infers failure. We ensure termination of the generic constraint propagation algorithm because the *cdf*-domain ordering is reflexive, antisymmetric and transitive.

Example 1. The result of adding two p-box *cdf*-intervals $\mathbf{I} \in [(2, 0.1, 7), (5.0, 0.8, 2.5)]$ and $\mathbf{J} \in [(3.0, 0.2, 1), (6, 0.3, 0.5)]$, $X =_{\mathcal{U}} \mathbf{I} +_{\mathcal{U}} \mathbf{J}$ is intersected with $X \in \mathcal{U}$; the domain of X accordingly will be pruned: $X \in [(5.0, 0.16, 0.75), (11, 0.4, 0.4)]$. When X, $\mathbf{I}$ and $\mathbf{J}$ are specified by one approximated uniform distribution, addition constraint yields $X \in [(5.0, 0.048), (11.0, 0.96)]$. Note that in the absence of *cdf* knowledge distribution, where candidate intervals have equal uncertainty weights, we obtain a real interval arithmetic addition.

Intuition. The steepness of the slope in both *cdf*-interval representations is an indication of how data is likely to be located within the interval. With a steeper slope data is more likely to occur towards lower quantiles of the interval and vise versa. p-box *cdf*-intervals representation adds-up more information to guide the decision maker. Risk averted is more likely to check the lower bound distribution of the p-box. This bound has the steepest slope. The upper *cdf*-bound aims at guiding the knowledge for a more risky decision making.

Consider in Example 1 the range of *cdf*-values $[0.048, 0.96]$ resulting from the addition yields a slope 0.152 hence data is equally probable over the range

of quantiles $[5, 11]$ with an average probability distribution 15.2%. On the other hand the p-box *cdf*-intervals of the addition is $X \in [(5.0, 0.16, 0.75), (11, 0.4, 0.4)]$. The p-box representation ranges the set of *cdf*-distributions by their stochastic dominance. This arrangement guides the decision maker such that for high risk aversion decision the lower bound triplet is taken under consideration because it provides the maximum possible probability data can occur and it is the most dominated probability bound. Due to its uniform property the lower bound probability reaches 100% at lower quantiles (in this example at 6.12) hence it can further prune the interval quantile range on the lower bound (i.e. when the *cdf* value is at 100% which yields a quantile range $[5, 6.12]$ for the running example). In addition the slope of the distribution quantifies the probability over the quantile range (75% on average). Observably the convex model output for this example is $[5, 11]$; it shows an equally weighted real interval with no information about the data chance of occurrence over the whole range of quantiles.

Global Linear Constraint Satisfaction. For linear constraints, we have embodied a global constraint satisfaction algorithm in the solver. p-box *cdf*-interval linear global constraint satisfaction aims at solving a system of linear equations which has p-box *cdf*-intervals coefficients and unknown variables. The algorithm is composed of five main steps:

1. Generate linear inequalities
2. Extract Interval Linear System (ILS) quantiles
3. Solve 2n eplex instances
4. Project the *cdf*-distributions
5. Extract the final solution p-box *cdf*-interval bounds

The algorithm prunes variable domains at fixed point using inference rules on the system of linear equations. The p-box *cdf*-intervals attached to the data (here coefficients) are propagated onto the *cdf*-variables. Inferences on the quantile component yield pruning on the resulting variable real domains and the additional information coming from the *cdf* and slope components demonstrate the information gained on the density of occurrence for the resulting points within the p-box *cdf*-domains.

Example 2. Consider the system of linear equations $(\mathbf{A}, \Re, \mathbf{b})$ shown below:

$$\mathbf{A} = \begin{bmatrix} [(1.0, 0.95, 0.41), (2.0, 0.41, 0.095)] & [(0.0, 0.83, 0.32), (2.0, 0.32, 0.083)] \\ [(0.0, 0.31, 0.75), (1.0, 0.75, 0.031)] & [(0.0, 0.35, 0.87), (1.0, 0.87, 0.035)] \\ [(1.5, 0.14, 0.96), (3.0, 0.96, 0.034)] & [(5.999, 0.9, 0.86), (6.0, 0.6, 0.028)] \end{bmatrix},$$

$$\Re = \begin{bmatrix} \subseteq_{\mathcal{R}_\mathcal{U}} \\ = \\ = \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} [(3.0, 0.88, 0.4), (4.0, 0.4, 0.088)] \\ [(0.0, 0.3, 0.78), (5.0, 0.78, 0.03)] \\ [(4.0, 0.29, 0.85), (15.0, 0.85, 0.029)] \end{bmatrix}$$

Fig. 2 illustrates two boxes, each encloses the output solution population residence for $X1$ and $X2$. The shown boxes are the result of applying p-box *cdf*-intervals propagation techniques on the linear equations provided in this example. Note that for the original *cdf*-intervals with an approximated uniform

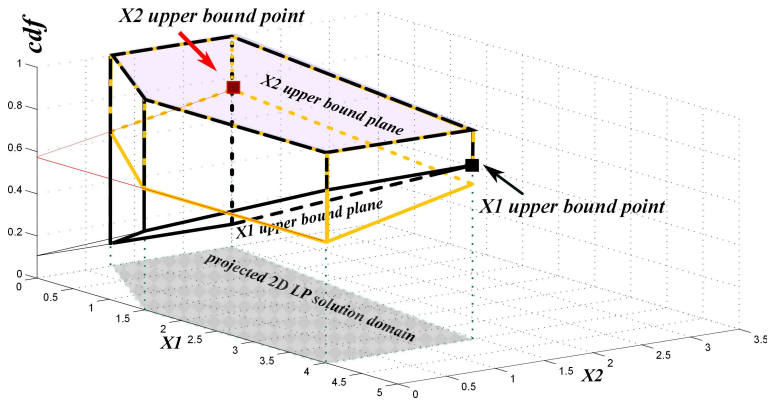


Fig. 2. Example 2: Solution set resulting from the p-box *cdf*-interval computations

distribution solution to the problem yield one plane per variable that is enclosed by the shown p-boxes and which specifies an approximated cummulative uniform distribution of the data whereabouts. The black box is the representation of $X1$ solution p-box domain $[(0.0, 1.0, +\infty),\ (4.0, 0.667, 0.1669)]$ and the yellow box is the solution domain $X2 \in (0.0, 1.0, +\infty),\ (2.5, 0.598, 0.4344)]$. Clearly, solutions intersect in the $X1$-$X2$ 2D space as illustrated by their projection depicted by the shaded checkerboard region: the real-interval arithmetic solution. This indicates that $X1$ and $X2$ quantiles range respectively between $[0.0, 4.0]$ and $[0.0, 2.5]$. The p-box representation shows a minimum possible chance of occurrence for quantiles respectively for $X1$ and $X2$ to be at most 4.0 is 66.7% and 2.5 is 59.8%. Note that quantiles of $X1$ and $X2$ cannot lie outside the p-box bounds defined by the two dimensions: quantiles and *cdf*. Additionally, for the most risky choice that elects the dominant probability distribution a more pruned quantile ranges respectively for $X1$ and $X2$ are $[0.0035, 4.0]$ and $[1.12, 2.5]$ with average probability distributions 16.7% and 43.44%.

4 Discussion and Conclusion

In summary, *cdf*-intervals introduced in [6] extend convex modeling that is favored to be computationaly tractable. The framework adds a quantitative information while seeking a linear approximation of the data distribution; we have augmented the framework with p-box bounds to enclose the whole collected data and its degree of knowledge specified by its unknown probability distribution.

Today we have a CP solver that seeks the realization of possible network flows in the network traffic analysis problem; fluctuating demands are input as coefficients to 3 solvers implemented in the ECL^{i}PS^e environment: 1) Uncertain CSP [9] (reliable modeling technique), 2) *cdf*-intervals [6], 3) p-box *cdf*-intervals.

Our empirical study shows that propagation in p-box *cdf*-intervals take almost the same time as UCSP to exploit variable solution domains; whereas time taken by *cdf*-intervals was almost doubled since the framework uses the same technique to implement the propagation once on each dimension: quantile and *cdf*. Indeed our goal is to add expressiveness to the solution sets of reliable models while preserving tractability. The choice of linear enclosure of the data distribution ensures both. The defined global linear constraint satisfaction algorithm runs 2 Simplex per variable to compute new interval bounds; this is commonly used in Interval Linear System and is very cost effective. We show in our case that bounding a random variable distribution, with two tight linear uniform *cdf* distributions, is a safe enclosure that adds quantitative information to the solution set produced, and does so at minimal overhead: a preprocessing step transforms our model into an ILS in O(2m) where m is the number of problem constraints, and we project *cdf*-distributions once the Simplex runs are completed to output the new p-box bounds.

Future work will cater the solver for applications with uncertain data. Measured data that is already defined in the problem definition thereof needs to be represented as *cdf*-interval coefficients and input to the solver. Hence solution sets will obtain an additional quantitative information which will show an adequate knowledge of data whereabouts. Applications like management of inventory with stochastic demand [7] are subject to be studied in the near future.

References

1. A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1–14, 1999.
2. J. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.
3. ECRC. Eclipse (a) user manual, (b) extensions of the user manual. Technical report, ECRC, 1994.
4. H. Fargier, J. Lang, and T. Schiex. Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proceedings of the National Conference on Artificial Intelligence*, pages 175–180. Citeseer, 1996.
5. A. Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1):99–118, 1977.
6. A. Saad, C. Gervet, and S. Abdennadher. Constraint Reasoning with Uncertain Data Using CDF-Intervals. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 292–306, 2010.
7. S. Tarim and B. Kingsman. The stochastic dynamic production/inventory lot-sizing problem with service-level constraints. *International Journal of Production Economics*, 88(1):105–119, 2004.
8. R. Williamson and T. Downs. Probabilistic arithmetic. I. Numerical methods for calculating convolutions and dependency bounds* 1. *International Journal of Approximate Reasoning*, 4(2):89–158, 1990.
9. N. Yorke-Smith and C. Gervet. Certainty closure: Reliable constraint reasoning with incomplete or erroneous data. *ACM Transactions on Computational Logic (TOCL)*, 10(1):3, 2009.
10. K. Zhou, J. C. Doyle, and K. Glover. *Robust and optimal control*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

Empirical Evaluation of AND/OR Multivalued Decision Diagrams for Inference

Student: William Lam, Advisor: Rina Dechter

Donald Bren School of Information and Computer Sciences
University of California, Irvine, CA 92697, USA
{willmlam,dechter}@ics.uci.edu

Abstract. AND/OR Multi-valued Decision Diagrams (AOMDD) were shown to provide a more compact representation of discrete-domain real-valued functions compared to other decision diagram variants [**?**]. We show the performance of AOMDDs on inference tasks in graphical models. We introduce the elimination operator to AOMDDs, which in conjunction with the combination operator introduced in previous work, yields a full bucket elimination (BE) scheme using AOMDDs as an alternative function representation to tables. We show that we are able to solve instances that do not fit in main memory when using tables.

1 Introduction

AND/OR Multi-valued Decision Diagrams (AOMDDs) combine the two frameworks of AND/OR search spaces and multi-valued decision diagrams (MDDs) to create a framework that compactly represents discrete-domain functions such as those in discrete graphical models [**?**]. The AND/OR search space is a more compact search space for search-based inference algorithms in graphical models compared to OR search spaces. For problems with decomposition into subproblems, the AND/OR search space captures this. Decision diagrams are generally used to represent functions compactly [**?**].

The key algorithm for combining AOMDDs, *apply*, first introduced in [**?**] was never implemented before. Our work also extends upon previous work by introducing the elimination operator to AOMDDs. With these two algorithms in place, this yields the full bucket elimination [**?**] scheme using AOMDDs as an alternative function representation to tables. We provide the first empirical results demonstrating the algorithm and contrasting its performance with the BE algorithm using tables.

Similar work is presented in [**?**,**?**], where an algebraic decision diagram (ADD) structure is considered. In [**?**], ADDs are extended with affine transformations to capture additive and multiplicative structures in graphical models. However, AND structure is still not exploited in these alternative decision diagram variants and they are restricted to variables with binary domains.

We start with presenting preliminaries by defining graphical models and counting queries.

Definition 1 ***(graphical model/counting query)*** *A graphical model is a tuple* $\mathcal{R} = \langle X, D, F, \otimes \rangle$*, where* $X = \{X_1, ..., X_n\}$ *is a set of variables,* $D = \{D_1, ..., D_n\}$ *is the set of the respective finite domains of the variables in* X*,* $F = \{f_1, ..., f_r\}$ *is a set of real-valued functions defined over a subset of variables* $S_i \subseteq X$*, and* $\otimes$ *is a combination operator (i.e.* $\prod, \sum, \bowtie$*). The graphical model represents a global function computed by* $\otimes_{i=1}^{r} f_i$*. For a CSP, the number of solutions is the number of assignments which do not violate any constraints. For a weighted CSP, the weighted solution count is the sum of the weights of all solutions such that no constraint is violated (having a weight of 0). For graphical models representing probability distributions, this is likelihood/ partition function computation. Formally the task is to find* $\sum \prod_{i=1}^{r} f_i$

We refer the reader to previous work for background on AND/OR search spaces and decision diagrams [**?**,**?**]. The basic idea of AOMDDs is augmenting context-minimal AND/OR search graphs to remove redundant nodes (or equivalently, augmenting weighted MDDs with AND nodes.) Overall, AOMDDs exploit determinism and context-specific independence [**?**] to achieve compactness. More details on AOMDDs are in [**?**].

2 Algorithms

In this work, we include the reduction rule by redundancy and use AOMDDs as an alternative to a tabular representation of the functions and messages in bucket elimination. To perform this, we require a method of applying the combination operator to AOMDDs and the elimination operators. For AOMDDs, it is presented here for the first time.

The main operation to perform the combination of two AOMDDs is the *apply* operator. It is stated that the runtime of apply is quadratic in the size of the input AOMDDs. We omit the full details of the algorithm for space issues and refer the reader to previous work [**?**].

There are difference and restrictions of the operation when compared to decision diagrams without AND decomposition. Since AOMDDs further compress a function representation by taking advantage of decomposition in the pseudo tree [**?**], operations on it are bound by the same rules as variable elimination. Namely, once we consider a fixed variable ordering, eliminating a variable whose children are not yet eliminated would induce edges in the induced graph between all of its neighbors. Equivalently, this means we would be changing the order of sum and product operators.

A basic description of the algorithm is as follows. From the embedded pseudo tree of the AOMDD, we create a list of *relevant* variables by tracing a path from the leaf node representing the elimination variable to the root of the tree. This creates a direct path from the root of the AOMDD down to the elimination variable without the need to explore other branches of the AOMDD. We then create a reverse BFS ordering based on list of relevant variables. If the node is an elimination variable, we eliminate the node by performing the necessary operator

and promote the weight to the parent. Otherwise, we normalize the node (making its AND children weights sum to 1) and pass on the normalization constant to the parent.

One caveat to note is that the metanode for a variable we are eliminating may not be present in the decision diagram due to the reduction rules. This is an issue when the elimination operator is summation. We must compensate for any missed metanodes, which we can identify if we see an ancestor of the elimination variable connected to a terminal metanode. Since nodes would be missing only if it were redundant, we can multiply the weight of that ancestor metanode by the domain size of the elimination variable. However, in the process after eliminating a node, the intermediate structure looks identical to that of when the input diagram does not have the node due to redundancy reduction. Therefore, we keep track of which nodes already received a weight from a child node to distinguish between these two.

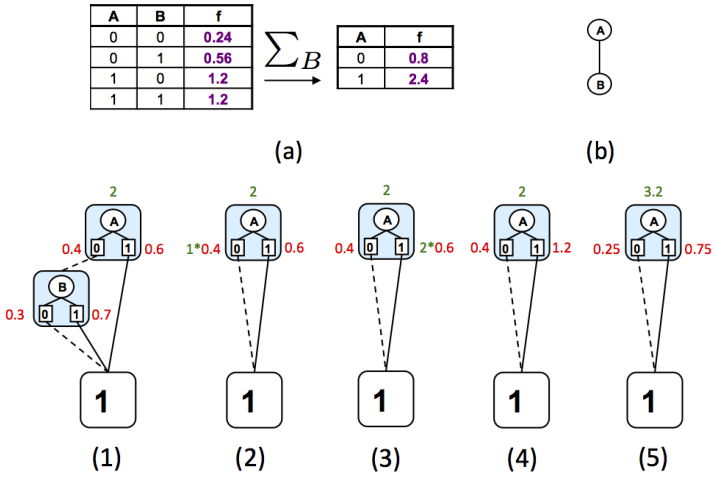


Fig. 1. Example of elimination on AOMDDs. The state of the AOMDD is shown through the process.

We demonstrate the algorithm on a small example, shown in Figure **??**. The function tables above (a) the AOMDDs demonstrate the operation performed in a standard representation. We are interested in summing out variable B. The embedded pseudo tree (b) is used to determine the set of relevant variables, which in this case is $\{A, B\}$.

We begin with the AOMDD shown at (1), which represents the same function as the input table. Visiting the relevant nodes in a reverse BFS order, we visit the metanode B first, eliminate it, and propagate its result up to the parent AND node in metanode A, shown in (2). At (3), we are left with only metanode A. Checking the 0 AND node, since it received a weight from something, we are

done with it. Checking the 1 AND node, since it has not received a weight, we multiply it by the domain size of B, which is 2 in this case. The result is now shown at (4). Finally, we normalize the AND node weights of metanode A and propagate its normalization term up to the root, yielding the resulting AOMDD in (5), which represents the same function as the output table.

In the cases of maximization and minimization, we do not encounter the same problem since these operators choose one from the set of values, which has no effect on functions where all the output values are identical. Conditioning can also be considered a form of elimination and also does not suffer from the issues that summation encounters for the same reasons.

With an elimination operator, this yields a full BE algorithm for inference using AOMDDs as a function representation (AOMDD-BE). The complexity remains the same as standard BE, as in the worst case, the AOMDD has as many AND nodes as the number of entries in the table. However, for some problem structures, the AOMDD size can be far smaller than the table size.

3 Experiments

For all tables in this section, for each problem instance, we report number of variables (n), induced width (w), height of the pseudo tree (h), maximum domain size (k), time, and memory usage. The algorithms were implemented in C++ (64-bit) and the experiments were run on 2.6 GHz machines with 24GB of RAM.

The following evaluates the AOMDD-BE algorithm, which is the same as bucket elimination, but uses AOMDDs to represent all functions. We ran experiments on the UAI 2006 evaluation problems and genetic linkage analysis networks, available at `http://graphmod.ics.uci.edu`. In each table, we compare the time and memory usages of standard BE vs. AOMDD-BE. Times reported as "OOM" indicate that the algorithm exceeded our memory bound. Results on memory usage are based on the usage of the cache storing nodes of the AOMDDs. For instances where BE runs out of memory, we simulated its execution by only passing information about scope sizes to compute the memory usage.

UAI 2006 benchmarks. Results are presented in Table **??**. In columns 5 and 6, we see the runtimes for BE and AOMDD-BE, while the last two columns show the the memory usages of BE and AOMDD-BE.

We see that our scheme is able to solve some problems which do not fit in standard main memory. These problems have structures that AOMDDs exploit well. Namely, the functions of these problems have many zero values that can be represented easily by AOMDDs. In addition, AOMDDs are able to take advantage of functions that have many values that are the same, but not necessarily zero. Such functions are present in a number of the instances on which it outperforms BE based on memory usage. However, there must be a significant amount of compression before we get any memory savings. Namely, as each node contains information to capture the structure of the problem, it means that much more memory is used when representing a function which has many different values.

problem	n	w	h	k	time (s) [BE]	time (s) [AOMDD-BE]	Mem (MB) [BE]	Mem (MB) [AOMDD-BE]
BN_22	2425	5	575	91	1	13	**26.93**	581.27
BN_28	24	5	9	10	1	13	**1.79**	568.36
BN_30	1156	48	179	2	OOM	38	1.50E+10	**245.93**
BN_32	1444	56	219	2	OOM	4384	4.45E+12	**3006.08**
BN_34	1444	55	220	2	OOM	145	2.30E+12	**515.45**
BN_40	1444	55	235	2	OOM	91	1.82E+12	**322.76**
BN_42	880	23	54	2	21	2	314.04	**21.62**
BN_46	499	22	49	2	18	<1	248.97	**1.99**
BN_49	661	44	59	2	OOM	1188	7.83E+08	**2991.78**
BN_53	561	48	95	2	OOM	4063	8.43E+09	**3303.48**
BN_61	667	44	61	2	OOM	17	9.46E+08	**235.72**
BN_65	440	61	95	2	OOM	1062	Overflow*	**2843.65**
BN_84	360	20	24	2	4	22	**24.76**	546.21
BN_92	422	22	33	2	26	23	**187.43**	433.65

name	n	w	h	k	time (s) [BE]	time (s) [AOMDD-BE]	Mem (MB) [BE]	Mem (MB) [AOMDD-BE]
pedigree1	334	15	61	4	2	14	**23.61**	210.09
pedigree9	1118	25	137	7	550	5301	7499.77	**4030.34**
pedigree18	1184	19	102	5	7	200	**136.13**	959.28
pedigree20	437	21	58	5	131	291	1393.90	**1030.66**
pedigree23	402	20	58	5	19	52	**241.57**	532.46
pedigree25	1289	23	86	5	146	1284	**2037.69**	2999.84
pedigree30	1289	20	102	5	13	307	**220.63**	1044.76
pedigree33	798	24	116	4	347	883	4277.26	**1368.42**
pedigree37	1032	20	62	5	OOM	3535	251109.68	**7992.43**
pedigree38	724	16	67	5	OOM	2201	172249.65	**6253.16**
pedigree39	1272	20	83	5	46	400	**772.20**	1555.68
pedigree44	811	24	79	4	516	3795	6153.63	**4782.29**

Table 1. UAI 2006 benchmarks and pedigree networks. For pedigree networks, instances not shown here (7,13,19,31,34,40,41,42,50,51) run out of memory with both algorithms. (* The size in MB could not be stored within a double precision number representation.)

We generally see that for lower treewidth networks, standard BE is sufficient and has better runtime, however, it is unable to solve problems with higher treewidth due to lack of memory.

Pedigree networks. We also ran experiments on genetic linkage analysis networks (known as pedigree), for which the partition function value of many of them were not known before the work in [**?**], which makes use of hard disk to push the memory restrictions of solving a problem.

The results are shown in Table **??**. As with the previous set or problem instances, timing results are shown in columns 5 and 6 while memory usage is shown in columns 7 and 8.

Our results are less promising on these networks. There are only two instances which AOMDD-BE manages to perform very well on, which standard BE would require about 30 times the amount of memory. For the rest that AOMDD-BE managed to solve, a large number of problems were solvable by standard BE with a shorter amount of time and less memory. Even with those where AOMDD-BE uses less memory, the runtime is often much worse, due to overhead in maintaining the properties of a canonical AOMDD. We can attribute these results this set of problems having overall less determinism and context-specific indepen-

dence. However, these results also demonstrate the use of decision diagrams on non-binary networks for inference, when compared to related work using ADDs [**?**,**?**].

4 Conclusion

For many hard problems (such as the pedigree networks), the overhead of using the minimal AOMDD structure for function representation actually results in worse performance in terms of both time and space. On other problems, such as the ISCAS networks in the UAI 2006 evaluation set, our scheme shows good performance despite having high treewidth. We demonstrated results reinforcing the potential of using AOMDDs for the classic BE algorithm. Future work would include comparing with related techniques exploiting determinism and context-specific independence such as ACE [**?**].

References

1. Mateescu, R., Dechter, R., Marinescu, R.: AND/OR multi-valued decision diagrams (AOMDDs) for graphical models. Journal of Artificial Intelligence Research **33**(1) (2008) 465–519
2. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8) (1986)
3. Mateescu, R., Dechter, R.: Compiling constraint networks into AND/OR multi-valued decision diagrams (AOMDDs). In: Principles and Practice of Constraint Programming (CP 2006). (2006) 10.1007/11889205_25.
4. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence **113**(1) (1999) 41–85
5. Chavira, M., Darwiche, A.: Compiling bayesian networks using variable elimination. In: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07). (2007) 2443–2449
6. Sanner, S., McAllester, D.: Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05). Volume 19. (2005) 1384
7. Boutilier, C., Friedman, N., Goldszmidt, M., Koller, D.: Context-specific independence in bayesian networks. In: Proc. of the 12th International Conference on Uncertainty in Artificial Intelligence (UAI-96). (1996) 115–123
8. Kask, K., Dechter, R., Gelfand, A.: BEEM: Bucket elimination with external memory. In: Proc. of the 26th Annual Conference on Uncertainty in Artificial Intelligence (UAI-10). (2010) 268–276
9. Gogate, V., Domingos, P.: Approximation by quantization. In: Proc. of the 27th Annual Conference on Uncertainty in Artificial Intelligence (UAI-11). (2011) 247–255
10. Chavira, M., Darwiche, A.: Compiling bayesian networks with local structure. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05). Volume 19. (2005) 1306

Practical Tractability of CSPs by Higher Level Consistency and Tree Decomposition

Shant Karakashian, Robert Woodward, and Berthe Y. Choueiry

University of Nebraska-Lincoln
{shantk|rwoodwar|choueiry}@cse.unl.edu

Abstract. One fundamental research result in the area of Constraint Processing (CP) is a condition that guarantees problem tractability by relating the consistency level of a Constraint Satisfaction Problem (CSP) to the structure of the problem. In our research, we propose to build effective problem-solving strategies that exploit the above-mentioned result in practice. To this end, our investigations target two fundamental mechanisms in CP: (1) Consistency properties and their algorithms and (2) Backtrack search in a tree decomposition. In particular, we propose a new consistency property whose level is controlled by a parameter, and present algorithms for enforcing it. Then, we investigate strategies for backtrack search that apply those algorithms in localized contexts defined by a tree decomposition of the constraint network.

1 Introduction

Research in Constraint Processing (CP) has identified various *islands of tractability* as classes of CSPs that are solvable in polynomial time in the size of the input. We single out the tractability condition specified by a relationship between the *level of a consistency* of a CSP and a structural parameter of the corresponding constraint network such as the *treewidth* or the *hypertree width*. The larger the width of the network is, the higher the level of consistency may need to be established in order to guarantee backtrack-free search. This approach is hindered in practice by two main difficulties: finding the treewidth or hypertree width of a constraint network is an $\mathcal{NP}$-hard task [1], and enforcing higher levels of consistency may require the addition of constraints to the CSP, thus modifying its structure and width parameters.

The question that we address in our research is: *How close can we approach in practice the tractability guaranteed by the relationship between the level of consistency in a CSP and the width of its constraint network?* We propose to achieve "practical tractability" by (1) proposing new local consistency properties whose level is controlled by a parameter and designing algorithms for enforcing them that do not modify the structure of the constraint network, (2) enforcing such consistency properties on the clusters of a tree decomposition of the CSP, and (3) adding redundant constraints in the separators between clusters to boost propagation and enhance communications between clusters.

The main algorithms that are used in practice for enforcing consistency consider combinations of at most three variables or two relations. The consistency properties proposed so far that apply to larger combinations of variables or constraints may in general require the addition of new constraints [2, 3]. Moreover, different problems require different levels of consistency. For this reason, it becomes important to explore new properties whose level of consistency can be controlled (i.e., parameterized consistency), but that do not modify the structure of the constraint network, and thus, do not increase its width.

The main techniques that exploit the structure of the constraint network for solving the CSP use a tree-decomposition embedding of the constraint network. Because finding the optimal decomposition is $\mathcal{NP}$-Hard, heuristics are used to find a 'good' decomposition such as join tree [4], hinge decomposition [5], and hypertree decomposition [6]. Moreover, synthesizing and storing a global constraint on the separators is necessary to guarantee backtrack-free search, but it is prohibitive in practice.

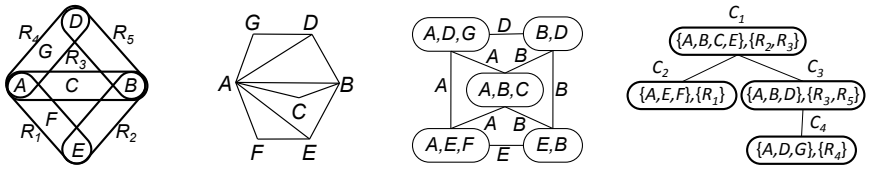
We propose to exploit the tree decomposition in the following problem-solving operations: (1) Localize the application of the consistency algorithms to the subproblems induced by the vertices of the tree decomposition; (2) Order the constraint-propagation process along the branches of the tree while favoring the most constrained paths; and (3) Enhance propagation by adding properly chosen redundant constraints between connected tree-vertices.

This paper is structured as follows. In Section 2, we give some necessary background information. In Section 3, we present the relational consistency property and outline two algorithms for enforcing it. In Section 4, we describe how we exploit the tree decomposition. Finally, in Section 5 we present some preliminary experimental results and conclude in Section 6. Preliminary results of parts of our work have already appeared in [7, 8].

2 Background

A constraint satisfaction problem (CSP) is defined by $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X}$ is a set of variables, $\mathcal{D}$ is a set of domains, and $\mathcal{C}$ is a set of constraints. Each variable $A_i \in \mathcal{X}$ has a finite domain $D_i \in \mathcal{D}$, and is constrained by a subset of the constraints in $\mathcal{C}$. Each constraint $C_i \in \mathcal{C}$ is defined by a relation R_i specified over the *scope* of the constraint, *scope*(C_i), which are the variables to which the constraint applies, as a subset of the Cartesian product of the domains of those variables. The *arity* of a constraint is the cardinality of its scope. A tuple $t_i \in R_i$ is thus a combination of values for the variables in the scope of the constraint that is either allowed (i.e., support) or forbidden (i.e., conflict). In this paper, we consider only allowed tuples. A solution to the CSP is an assignment, to each variable, of a value taken from its domain such that all the constraints are satisfied. Solving a CSP consists in finding one or all solutions.

A CSP can be represented by several types of graphs: in the *hypergraph* of a CSP, as shown in Fig. 1a, the vertices represent the variables of the CSP and the hyperedges represent the scopes of the constraints. The *primal graph* of a CSP is



(a) (b) (c) (d)

Fig. 1: (a) Hypergraph, (b) Primal graph, (c) Dual graph, (d) Tree decomposition

a graph whose vertices represent the variables and the edges connect every two variables that appear in the scope of some constraint as shown in Fig. 1b. The *dual graph* of a CSP is a graph whose vertices represent the constraints of the CSP, and whose edges connect two vertices corresponding to constraints whose scopes overlap as in Fig. 1c. The dual CSP, $\mathcal{P}_D$, is thus a binary CSP where: (1) variables are the constraints of the original CSP; (2) the variables' domains are the tuples of the corresponding relations; and (3) the constraints enforce *equalities* over the shared variables. A tree decomposition of a CSP is a tree embedding of the constraint network of the CSP. The tree nodes are thus *clusters* of variables and constraints. A tree decomposition must satisfy two conditions: (1) each constraint appears in at least one cluster and the variables in its scope must appear in this cluster, and (2) for every variable, the clusters where the variable appears induce a connected subtree. Fig. 1d shows a tree decomposition of the CSP in Fig. 1a. A *separator* of two adjacent clusters is the set of variables in both clusters. A given tree decomposition is characterized by its *treewidth*, which is the maximum number of variables in a cluster minus one.

3 Relational Consistency Property and Algorithms

We introduce the property R($*$,m)C as a relational consistency property for non-binary CSPs [7, 8]. This property ensures that, given any set of m constraints, every tuple in the relation of one of those m constraints can be extended to all the variables in the union of the scopes of the constraints in an assignment that simultaneously satisfies all the constraints. We present two algorithms for enforcing this consistency property on a CSP; both algorithms are based on solving by backtrack search the dual CSP induced by the m relations:

- PerTupleSearch considers each tuple in each one of the m relations and ensures, by backtrack search, that it appears in a solution to the induced dual CSP. Thus, it solves, in the worst case, as many satisfiability problems as there are tuples in the m relations.
- AllSolSearch finds all the solutions of the induced dual CSP to determine which tuples must be kept. It executes a single backtrack search but may have to find, in the worst case, all the solutions of the induced CSP.

4 Relational Consistency on Tree Decomposition

We investigate the use of R$(*, m)$C in the context of a tree decomposition for the purpose of localizing the application of the consistency algorithms to the subproblems induced by the vertices of the clusters, ordering the constraint-propagation process along the branches of the tree, and enhancing propagation by adding redundant constraints to the separators.

Localizing Relational Consistency: Instead of computing the combinations of m constraints over the entire CSP, we propose to restrict ourselves to the combinations computed within each cluster, thus reducing the number of combinations to be considered.

Ordering the Propagation: We consider two strategies to order the clusters in which the consistency algorithm is applied. The first strategy follows the fixed order of clusters given by the MaxCliques algorithm [9]. The second strategy prioritizes the clusters by favoring those clusters whose children are most constrained. We evaluate the constraintedness of a cluster by the ratio of the removed tuples to the original tuples in its relations.

Redundancy at Separators: The application of R$(*, m)$C to a set of relations is always followed by a step where the filtered constraints are projected on the domain of the variables. When applying R$(*, m)$C individually to each cluster of a tree decomposition, the effects of filtering in one cluster are transferred to the adjacent cluster through the domains of the variables in the separator between the two clusters. Enforcing R$(*, m)$C does not require adding new constraints to the CSP. However, synthesizing a global constraint at each separator improves the 'communication' between clusters and guarantees backtrack-free search.

Synthesizing and storing those global constraints is typically prohibitive, especially in terms of space. For this reason, we propose to approximate the global constraints by adding redundant constraints. The strategy that we adopt here consists in adding the clusters' constraints to the separator after projecting them on the variables in the separator. In Section 6, we describe two other strategies.

5 Experimental Results

The experiments reported below evaluate some of the techniques described in this paper. We generate a tree decomposition that is an adaptation of the the tree-clustering technique of [4], by building the primal graph of the non-binary CSP, triangulating it, then using the join tree of the maximal cliques of the resulting triangulation. We add redundant constraints to the separators by projecting the constraints in the cluster on the variables in the separator. We use the PerTupleSearch algorithm localized to the clusters of a tree decomposition. The parameter m of R$(*, m)$C is set to the number of relations in a cluster, and, thus, the *consistency level enforced adapts locally to each cluster* in the tree

Table 1: Comparing R$(*, m)$C localized to the clusters with propagation orderings Priority and MaxClique against GAC and maxRPWC. Averages are computed over instances completed by all methods.

	#	Completed				#BT Free				Avg CPU in seconds				Avg #NV			
		Priority	MaxClique	GAC	maxRPWC	Priority	MaxClique	GAC	maxRPWC	Priority	MaxClique	GAC	maxRPWC	Priority	MaxClique	GAC	maxRPWC
aim-100	24	20	**21**	15	16	15	**16**	1	1	18.66	**15.82**	509.02	479.67	**103.36**	**103.36**	10M	7M
aim-200	24	**9**	7	8	8	**8**	7	0	0	-	-	-	-	-	-	-	-
aim-50	24	24	24	24	24	**21**	**21**	1	3	1.01	**0.95**	1.51	1.31	**53.21**	**53.21**	43K	31K
comp-25	50	**45**	**45**	10	10	**44**	43	0	0	821.32	1,124.00	**719.61**	866.17	**45.29**	**45.29**	1M	1M
comp-75	40	**38**	37	3	3	**38**	37	0	0	8.19	4.54	**0.13**	0.17	**0.00**	**0.00**	1.00	1.00
dag-rand	25	**25**	**25**	5	0	**25**	**25**	0	0	-	-	-	-	-	-	-	-
ehi-85	100	**91**	10	74	55	**91**	10	0	0	417.65	450.23	**369.07**	663.08	**0.00**	**0.00**	86K	86K
ehi-90	100	**98**	7	55	44	**98**	7	0	0	383.21	539.02	**0.64**	0.91	**0.00**	**0.00**	10.00	10.00
modRenault	50	**50**	**50**	25	32	**50**	**50**	5	18	**5.91**	6.17	64.89	209.49	**81.39**	**81.39**	341K	1,213.30

decomposition. The propagation is ordered by the two strategies proposed in Section 4. Thus, we use two configurations for R$(*, m)$C and we refer to them as MaxClique and Priority.

We compare the two configurations of R$(*, m)$C against GAC2001 [10] and maxRPWC [11]. The four consistency algorithms are integrated as full lookahead strategies in a backtrack search using the domain/degree heuristic for dynamic variable ordering. The nodes in the search tree correspond to instantiations of the original variables of the CSP, and the count of node visits is the same for the four compared techniques. The experiments are conducted on benchmarks from the CSP Solver Competition[1] that are difficult to solve using GAC. We imposed a time limit of one hour per instance.

Table 1 gives the total number of instances in each benchmark, and the number of instances: solved within the time limit, and solved without backtracking. On the right side of the table, the average CPU time and number of nodes visited are given computed on the instances solved using the four algorithms.

We observe that R$(*, m)$C is able to solve most instances in those benchmarks without backtracking, thus achieving the practical tractability that we are aiming at. The average numbers of node visits show that R$(*, m)$C visits orders of magnitude fewer nodes than GAC and maxRPWC. The impact of fewer backtracking achieved by R$(*, m)$C is reflected in the number of instances completed in each benchmark. Also, we notice that by using the Priority ordering we are able to solve more instances than using the maxClique ordering.

6 Conclusion and Future Work

We propose a new local consistency property R$(*, m)$C with algorithms for enforcing it by localizing the application of the algorithms to the clusters of a tree

[1] http://www.cril.univ-artois.fr/CPAI08/

decomposition of a CSP. We also propose to modify the structure of the CSP to enhance propagation without increasing the width of the constraint network. We presented the results of our preliminary experiments comparing R$(*, m)$C to GAC and maxRPWC. The results indicate that we can achieve tractability in practice on many instances by solving them almost without backtracking.

In the future, we will evaluate two other strategies for approximating the global constraints on the separators: (1) adding binary constraints to the separator by generating a constraint for every fill-in edge obtained by a triangulation of the primal graph of the separator and (2) adding non-binary constraints to the separator that cover the maximal cliques of a triangulation of the separator's primal graph. Finally, we will evaluate our approach to count the number of solutions of the CSP and compare it to the BTD [12].

Acknowledgments Experiments were conducted on the equipment of the Holland Computing Center at the University of Nebraska-Lincoln. This research is supported by NSF Grant No. RI-111795.

References

1. Arnborg, S.A., Corneil, D.G., Proskurowski, A.: Complexity of Finding Embeddings in a K-Tree. SIAM Journal on Algebraic Discrete Methods **8** (1987) 277–284
2. Freuder, E.C.: Synthesizing Constraint Expressions. Communications of the ACM **21 (11)** (1978) 958–966
3. Dechter, R., van Beek, P.: Local and Global Relational Consistency. Theoretical Computer Science **173**(1) (1997) 283–308
4. Dechter, R., Pearl, J.: Tree Clustering for Constraint Networks. Artificial Intelligence **38** (1989) 353–366
5. Cohen, D.A., Jeavons, P., Gyssens, M.: A unified theory of structural tractability for constraint satisfaction problems. Journal of Computer and System Sciences **74**(5) (2008) 721–743
6. Gottlob, G., Scarcello, F.: Hypertree decompositions: A survey. In: Proceedings of the 26^{th} International Symposium on Mathematical Foundations of Computer Science (MFCS 01). (2001) 37–57
7. Karakashian, S., Woodward, R., Reeson, C., Choueiry, B.Y., Bessiere, C.: A First Practical Algorithm for High Levels of Relational Consistency. In: Proceedings of the 24^{th} AAAI Conference on Artificial Intelligence (AAAI 10). (2010) 101–107
8. Karakashian, S., Woodward, R.J., Choueiry, B.Y., Bessiere, C.: Relational Consistency by Constraint Filtering. In: Proceedings of the 25^{th} ACM Symposium On Applied Computing (ACM SAC 10). (2010) 2073–2074
9. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press Inc., New York, NY (1980)
10. Bessiere, C., Régin, J.C., Yap, R.H., Zhang, Y.: An Optimal Coarse-Grained Arc Consistency Algorithm. Artificial Intelligence **165(2)** (2005) 165–185
11. Bessiere, C., Stergiou, K., Walsh, T.: Domain Filtering Consistencies for Non-Binary Constraints. Artificial Intelligence **172** (2008) 800–822
12. Jégou, P., Terrioux, C.: Hybrid Backtracking Bounded by Tree-Decomposition of Constraint Networks. Artificial Intelligence **146** (2003) 43–75

Embedded System Verification Through Constraint-Based Scheduling

Olfat EL-Mahi, Gabriela Nicolescu, Gilles Pesant and Giovanni Beltrame

Department of Computer and Software Engineering
École Polytechnique de Montréal, Québec, Canada
{olfat.ibrahim,gabriela.nicolescu,gilles.pesant,giovanni.beltrame}@polymtl.ca

Abstract. This work introduces Constraint Programming (CP) as a powerful tool for the verification of performance metrics of Multiprocessor System-on-Chip (MPSoCs). Our methodology was evaluated using streaming applications mapped onto a target MPSoC. The resulting constraint-based scheduling problem allowed us to identify performance constraint violations in a fraction of the time required by simulation-based verification.

1 Introduction

Multiprocessor System-on-Chip (MPSoC) designs have become a very popular choice for modern embedded systems [1]. These designs use complex on-chip networks to integrate different programmable processor cores, specialized memories, and other components on a single chip. The parallel nature of MPSoCs makes verification a challenging task, in particular for communication and multimedia applications. This is due to the non-functional constraints of hardware and software modules, such as processor speed, buffer size, energy budget, and scheduling policy [2], and the combination of multiple applications.

System-level design and verification methodologies such as Constraint Programming (CP) have been introduced as a solution to handle the design complexity of embedded systems [2]. The power of CP comes from the fact that validity, quality, and test specification requirements for any system are naturally modelled through constraints, which are naturally represented as a Constraint Satisfaction Problem (CSP). In this paper, we introduce a constraint-based scheduling model for concurrent streaming applications on MPSoCs. Our aim is to identify which critical system parameters (e.g. buffer size) can lead to unsatisfied application constraints.

This paper is structured as follows: the related work is introduced in Section 2; our MPSoC architecture platform is described in Section 3; Section 4 outlines our constraint based scheduling model and the associated constraint programming techniques; Section 5 shows our experimental results; finally, Section 6 draws some concluding remarks.

2 Related Work

Simulation-based verification is a well known method to determine the response time of embedded systems. Virtual platforms [3] can simulate both the software

and the hardware of a computer system in detail, at the expense of long execution times. At a higher level of abstraction, one can find scheduling simulators that only analyse the scheduling of the system's tasks based on key attributes and execution times. In general, scheduling problems are computationally challenging, and have been subject of active research in Constraint Programming (CP) and in Operations Research (OR) for many years [4]. Recently, hybrid approaches have been applied to this class of problems [5]: most of them split the overall problem into an assignment and a scheduling sub-part. Those are solved in an iterative and interactive fashion with a mix of CP and OR techniques.

Streaming applications like MPEG4 or VOIP define a pipeline work flow with strict ordering of data transfers between processing elements. Verification of these systems requires creating a system-level scenario [6]. Verification of embedded streaming applications on MPSoCs has been widely explored by using the Synchronous Data Flow (SDF) model [7], which presents general techniques to construct periodic admissible parallel schedules (PAPS) on limited number of processors. [2] further investigates buffer minimization and task scheduling issues for streaming applications. Some CP approaches for data-stream (or cyclic) scheduling and MPSoCs have been introduced [8].

Our work extends these previous by considering multi-stream models on MPSoCs using CP. To the best of our knowledge, no previous attempt exists to address these problems using constraint programming.

3 Architecture Platform

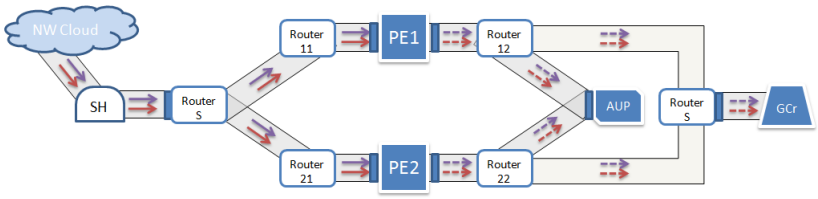


Fig. 1: MPEG-4 and VOIP packet flow in MPSoC architecture. SH = SharedMem, PE1 = Processor1, PE2 = Processor2, AuP = AudOP, GraphCrd = GCR

In this paper, we consider a regular 2-D mesh Network On Chip (NoC), usually scalable up to 64 tiles, as our Design Under Test (DUT). As a case study we used MPEG4 and VOIP applications, (see Figure 1). The system architecture consists of: one shared memory (SharedMem) acting as a receiver for different application streams, two processing elements each with its own private memory to generate the output stream, one graphics card for video display (GraphCrd), and an output port for audio display (AudOP), connected via 2x2 ST-NoC routers. The inputs to the system are packetized trace files for both applications. This makes the application more realistic and allows using different streams with different rates.

Table 1: Symbols used in the model

Symbol	Description	Equation
η	Packet Size in bytes	input
σ, ρ	number of Frames for Application M, V respectively	input
ϱ_i, υ_j	original frame size for Application M, V respectively where $0 \leq i \leq \sigma - 1$ and $1 \leq j \leq \rho - 1$	input
Γ, ϑ	Decompressed frame size for Application M, V respectively	input, $Max(\upsilon_i) * 15$
α, β	number of m, v packets respectively	$\sum_{i=1}^{\sigma}(\varrho_i/\eta), \sum_{i=1}^{\rho}(\upsilon_i/\eta)$
γ, δ	number of $m^{'}, v^{'}$ packets respectively	$(\Gamma/\eta) * \sigma, (\vartheta/eta) * \rho$
χ	number of Processors on the system	input
$\hat{t}$	number of tasks for m, v packets	input
$\hat{t}^{m^{'}}, \hat{t}^{v^{'}}$	number of tasks for $m^{'}, v^{'}$ packets respectively	input
$\hat{a}, \hat{a}^{'}$	$a, a^{'}$ chain length	input
$\hat{f}, \hat{f}^{m^{'}}, \hat{f}^{v^{'}}$	$f, f^{m^{'}}, f^{v^{'}}$ chain length	$\hat{t} - \hat{a}, \hat{t}^{m^{'}} - \hat{a}^{'}, \hat{t}^{v^{'}} - \hat{a}^{'}$

4 Constraint-Based Scheduling

We mapped streaming applications onto the target MPSoC architecture by modelling them as Constraint-based scheduling problems. Packets from these applications are translated to sets of tasks. The output either gives a suitable schedule for the input stream or it indicates that no solution exists.

Stream models Among standard types of scheduling problems, our problem is closest to flow shop scheduling, known to be NP-hard. The flow shop scheduling problem consists of a finite set of jobs to be processed on each of a finite set of machines. Jobs have the same processing order through the machines but the order in which uninterruptible jobs are processed on a given machine can vary between machines. Machines generally have a processing capacity and each job-machine pair has its own capacity demand and processing time.

Following the DUT in Figure 1 we have two streaming applications M and V (for MPEG4 and VOIP respectively), each with two sets of packet type (Original, Decompressed) processed through one of two different chains of operations depending on the output components and the chosen alternative resources. Original packets $P^m = \{m_i\}_{i=1}^{\alpha}$ and $P^v = \{v_r\}_{r=1}^{\beta}$ correspond to the streams received by the DUT into the shared memory and traveling through the system until being processed on one of the two processors. Decompressed packets $P^{m'} = \{m'_j\}_{j=1}^{\gamma}$ and $P^{v'} = \{v'_i\}_{s=1}^{\delta}$ correspond to the streams generated in the processor as the decompressed packets are the output of the original packets. They travel through the system until they reach the output resource element. We denote by $P = P^m \cup P^{m'} \cup P^v \cup P^{v'}$ the set of all packets. Each packet in each stream will be treated as a sequence of tasks, and each task is processed using a single resource. Additional constraints come from the system architecture and applications.

We will simplify our model based on two facts. First the scheduling test uses a trade-off between video quality (measured in terms of frame size) and buffer size. Second the most critical system resources are buffer capacity and processor frequency. Therefore the network interfaces (NI) and shared memory delays will be expressed by a minimum temporal separation between tasks processed on two consecutive components. Also shared memory size can be reasonably big for all

packets to stay there as long as they need without violating other constraints. The main symbols in the model are described in Table 1.

The DUT set of resources is $\{SharedMem = 1, RouterS = 2, Router_{11} = 3, Processor_1 = 4, Router_{12} = 5, Router_{21} = 6, Processor_2 = 7, Router_{22} = 8, Router_3 = 9, GraphCrd = 10, AudOP = 11, NI = 12\}$ — two properties are associated to each resource, speed and capacity.

A packet travelling through the system uses different resource chains from the receiver (shared memory then processed by one of the two processors then passed to its output device). The existence of more than one processor makes some of the resources alternatives. For example $\langle SharedMem, RouterS, Router_{11}, Processor_1\rangle$ and $\langle SharedMem, RouterS, Router_{21}, Processor_2\rangle$ are alternative resource chains for original packets before compression.

Here we identified the five possible different resource chain orders: $f = \{2\}$ for non alternative resources chain of applications original packets type, $a = \{\{3, 4\}, \{6, 7\}\}$ and $a' = \{\{4, 5\}, \{7, 8\}\}$ for alternative resources chain of applications original and decompressed packets type respectively, $f^{m'} = \{9, 10\}$ and $f^{v'} = \{11\}$ for non alternative resources chain of applications M, V decompressed packets type respectively.

Decision Variables Packets are represented by the union of four main sets of tasks T processed on different resource chains (See Figure 1 and Table 1 for details):

$$T = \bigcup_{0\le i\le \alpha-1} T^{m_i} \bigcup_{0\le j\le \gamma-1} T^{m'_j} \bigcup_{0\le r\le \beta-1} T^{v_r} \bigcup_{0\le s\le \delta-1} T^{v'_s} \tag{1}$$

where $T^{m_i} = \{t_k^{m_i} : 0 \le k \le \hat{t} - 1\}$, $T^{m'_j} = \{t_k^{m'_j} : 0 \le k \le \hat{t}^{m'} - 1\}$, $T^{v_r} = \{t_k^{v_r} : 0 \le k \le \hat{t} - 1\}$, and $T^{v'_s} = \{t_k^{v'_s} : 0 \le k \le \hat{t}^{v'} - 1\}$. We further refine tasks by associating one to each alternative resource in set aT:

$$aT = \bigcup_{0\le i\le \alpha-1} aT^{m_i} \bigcup_{0\le j\le \gamma-1} aT^{m'_j} \bigcup_{0\le r\le \beta-1} aT^{v_r} \bigcup_{0\le s\le \delta-1} aT^{v'_s} \tag{2}$$

where $aT^{m_i} = \{at_{kp}^{m_i} : \hat{f} \le k \le \hat{t} - 1, 0 \le p \le \chi - 1\}$, $aT^{m'_j} = \{at_{kp}^{m'_j} : 0 \le k \le \hat{a}' - 1, 0 \le p \le \chi - 1\}$, $aT^{v_r} = \{at_{kp}^{v_r} : \hat{f} \le k \le \hat{t} - 1, 0 \le p \le \chi - 1\}$, and $aT^{v'_s} = \{at_{kp}^{v'_s} : 0 \le k \le \hat{a}' - 1, 0 \le p \le \chi - 1\}$.

As usual to each task we associate *start*, *end* which are the time the packet starts and ends being processed on the corresponding component respectively, *demand* which is the space it occupies on the component while being processed by it, and *presence* which indicates whether the packet used a certain resource; for set T this value is equal to 1 whereas for aT this value is either 1 or 0.

Constraints The binding of streaming applications onto the target MPSoC architecture is a process with resource limitations and RT requirements. Here we use a constraint-based formulation to model the application-to-architecture mapping, communication routing, flow control, and computation scheduling. Basically we have four main sets of constraints controlling demand, duration, scheduling start and end time, and capacity (see Table 2). The constraints are chosen to ensure both system and application rules are respected.

Table 2: Model Constraints

To	Rule
all	1. resource capacity must be respected at all time.
all	2. packets must end processing before simulation deadline.
all	3. packet duration must be as large as the minimum duration a certain resource needs to process it.
m, v	4. packets must not start processing on the system before their arrival time.
all	5. decompressed packets of one frame cannot start before receiving all packets from its original frame.
all	6. original packets of one frame cannot end before starting all packets of its decompressed frames
$m, v, v^{'}$	7. all packets are processed in sequential order.
$m^{'}$	8. application M uses a periodic pattern known as a Group of Pictures (GOP) causes a difference in the sequence of data transmitted and data displayed: $m^{'}$ packets must follow order its frame order 1, 4, 2, 3, 7, 5, 6, 10, 8, 9, 1, 11, 12.
all	9. tasks for a given packet are processed in order.
all	10. alternative tasks must be processed on only one of the alternative resources
all	11. packets of the same frame must be processed on the same processor, and frames depending on other frames must be processed on the same processor
$m^{'}$	12. $m^{'}$ packets of the same frame must be displayed at the same time.
$v^{'}$	13. $v^{'}$ packets of the same frame must respect a display rate of 50 frames per second

5 Experimental Results

Our aim is to experimentally identify non-trivial cases of system failure which are unlikely to be detected manually by a Test Engineer. Particular interest is given to cases that show the impact of independent applications sharing the same computational resources. Our model was implemented using IBM ILOG OPL IDE v6.3 and used the default search. All experiments were run on an Intel Core i7 computer with 4GB RAM. The target parameters are shown in Table 3. The design space is explored by manually and sequentially applying these parameters.

Table 3: Design space for the experimental platform

Parameter	From	To	Parameter	Value
PE BW	1Mb/s	512Mb/s	Max Allowed VOIP Delay	1s
PE Size	1.5KB	15KB	Num. of PEs	2
Bus Latency	10ns	100ns	Memory BW	2 Gb
Bus Latency	10ns	100ns	Packet Size = Buffer Size	1.5 KB
Max Allowed VOIP Delay	2ms	1s	MPEG frames Size	$QCIF, SDTV$
Simulation dead line	2 seconds (Double the time needed to display frames)			

We take our results for application M (with two different frame sizes) and Application V (with either a delay restriction like a phone call, or some allowed buffering flexibility like voice message) separately and when combined.

Some tests gave straightforward results: both applications will always fail if the private memory of the PEs is less than 3 KB and 6 KB for M and V, respectively. This is due to inter- and intra-frame packet dependencies. Other tests show how one application failure might affect the other, when running simultaneously. For example, when M is dealing with an SDTV stream with a PE BW of 64Mb/s, the non-restricted delay version of V will always fail. Similarly, the delay-restricted version of V will always fail when M is processing a QCIF stream with a PE BW of 64Mb/s.

Our methodology has allowed us to identify some interesting cases. For example the combination of M and V will fail with a PE BW 11 Mb/s even if both applications can be successfully scheduled independently, meaning that our methodology can identify issues due to the non-obvious interaction of multi-

ple applications. Given that PEs need to be fast enough to produce the required frame rate, one can devise the following empirical rule: $\frac{F_s fps/P_s}{Out_{max}} > 1$ with F_s, P_s the frame and packet size, respectively, and Out_{max} the maximum processor output. Hence the DUT should fail when running only M using QCIF with a PE BW $<$ 14.5 Mb/s. Nevertheless our methodology shows that a PE BW = 11 Mb/s is sufficient for the application. This shows that non-trivial optimizations can be discovered with our methodology. Conversely, for V, a PE BW = 6.8 Kb/s should be sufficient, but we can observe that in delay-restricted conditions the system could not be scheduled if PE BW $<$ 64 Mb/s. This shows we can detect issues related to buffering and link delays.

To test the performance of our methodology, we compared it with the use of the ReSP MPSoC Simulation Platform [3]. When using FFMPEG, our system can detect system failure in less then 15 seconds and verify system success in 10 minutes, while ReSP takes around 30 minutes to run one simulation.

6 Future Work

Our next steps will be: improving performance by designing a dedicated search strategy, experimenting with different applications, and handling tasks differently depending on their nature (i.e. heavy tasks to be tested with CP and light tasks tested by simulation).

References

1. Wolf, W.: The future of multiprocessor systems-on-chips. In: Proceedings of the 41st annual Design Automation Conference, ACM (2004) 681–685
2. Zhu, J., Sander, I., Jantsch, A.: Constrained global scheduling of streaming applications on mpsocs. In: Proceedings of the 2010 Asia and South Pacific Design Automation Conference, IEEE Press (2010) 223–228
3. Beltrame, G., Fossati, L., Sciuto, D.: Resp: a nonintrusive transaction-level reflective mpsoc simulation platform for design space exploration. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **28**(12) (2009) 1857–1869
4. Lombardi, M., Milano, M.: Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. Constraints (2012) 1–35
5. Hooker, J.: A hybrid method for planning and scheduling. Principles and Practice of Constraint Programming–CP 2004 (2004) 305–316
6. Nahir, A., Ziv, A., Emek, R., Keidar, T., Ronen, N.: Scheduling-based test-case generation for verification of multimedia socs. In: Proceedings of the 43rd annual Design Automation Conference, ACM (2006) 348–351
7. Lee, E., Messerschmitt, D.: Static scheduling of synchronous data flow programs for digital signal processing. Computers, IEEE Transactions on **100**(1) (1987) 24–35
8. Bonfietti, A., Lombardi, M., Benini, L., Milano, M.: Global cyclic cumulative constraint. Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems (2012) 81–96

New spanning tree relaxations for the Asymmetric TSP

Jean-Guillaume Fages (student) and Xavier Lorca (supervisor)

June 9, 2012
École des Mines de Nantes,
LINA UMR CNRS 6241,
FR-44307 Nantes Cedex 3, France
{Jean-Guillaume.Fages,Xavier.Lorca}@mines-nantes.fr

Abstract. Recent works on cost based relaxations have improved Constraint Programming (CP) models for the Traveling Salesman Problem (TSP). In this paper, we suggest to refine existing tree based relaxations of the TSP by considering properties of the reduced graph.

1 Introduction

Given a n node, m arc complete directed weighted graph $G = (V, A, f : A \to \mathbb{R})$, the Asymmetric Traveling Salesman Problem [1] (ATSP) consists in finding a partial subgraph $G' = (V, A', f)$ of G which forms a Hamiltonian circuit of minimum cost. This NP-hard problem is one of the most studied by the Operation Research community. It has various practical applications such as vehicle routing problems, microchips production optimization or even scheduling.

The symmetric TSP is well handled by linear programming techniques [1]. However, such methods suffer from the addition of side constraints and asymmetric cost matrix, whereas constraint programming models do not. Since the real world is not symmetric and industrial applications often involve constraints such as time windows, precedences, loading capacities and several other constraints, improving the CP general model for solving the ATSP leads to make CP more competitive on real world routing problems. Recent improvements on cost based relaxations [2] had a very strong impact on the ability of CP technologies to solve the TSP. In this paper, we present how both the Minimum Spanning Tree (MST) and the Minimum Spanning Arborescence (MSA) relaxations can be improved by considering the reduced graph. This work is part of a more general study [6] where we investigate how the graph structure can contribute to the resolution process, in order to tackle larger instances.

2 Background

Let us consider a directed graph $G = (V, A)$. A *Strongly Connected Component* (SCC) is a maximal subgraph of G such that for each pair of nodes $\{a, b\} \in V^2$, a path exists from a to b and from b to a. A *reduced graph* $G_R = (V_R, A_R)$ of a

directed graph G represents the SCCs of G. This graph is obtained by merging the nodes of G which are in the same SCC and removing any loop. Such a graph is unique and contains no circuit. We link G and G_R with two functions: $\texttt{sccOf} : V \to V_R$ and $\texttt{nodesOf} : V_R \to V^V$.

In a CP context a *Graph Variable* [5, 10] can be used to model a graph. We define a graph variable GV by two graphs: the graph of *potential* elements, $G_P = (V_P, A_P)$, contains all the nodes and arcs that potentially occur in at least one solution whereas the graph of *mandatory* elements, $G_M = (V_M, A_M)$, contains all the nodes and arcs that occur in every solution. Thus, $GV = (G_P, G_M)$. It has to be noticed that $G_M \subseteq G_P \subseteq G$, where G is the (complete) input graph. It should also be noticed that, regarding the TSP, $V_P = V_M = V$, so the resolution will focus on A_M and A_P: branching strategies and propagators will remove infeasible arcs from A_P and add mandatory arcs of A_P into A_M.

3 A basic model

Given, a directed weighted graph $G = (V, A, f)$, and a function $f : A \to \mathbb{R}$, the ATSP consists in finding a partial subgraph $G' = (V, A', f)$ of G which forms a Hamiltonian circuit of minimum cost. A simple ATSP model in CP, involving a graph variable GV, can basically be stated as minimizing the sum of costs of arcs in the domain of GV and maintaining GV to be a Hamiltonian circuit. However, it is often more interesting to convert such a model in order to find a path instead of a circuit [7, 8]. Our motivation for this transformation is that it brings graph structure that is more likely to be exploited.

In this paper, we consider the ATSP as the problem of finding a minimum cost Hamiltonian path with fixed start and end nodes in a directed weighted graph. In the following, $s, e \in V$ respectively denote the start and the end of the expected path. s and e are supposed to be known. They can be obtained by duplicating any arbitrary node, but it makes more sense to duplicate the node representing the salesman's home. Then, an efficient filtering can be obtained with the `AllDifferent` constraint which maintains a node-successor perfect matching [9] and the `NoCycle` constraint [4]. This would however not be sufficient to solve big instances. Instead, the model should embed relaxation based constraints, to provide inference from costs. We will focus on the widely exploited MST relaxation [11], where both the degree constraint and arc direction are relaxed, and the MSA relaxation [4] where only the degree constraint is relaxed. While the MSA is clearly more accurate, the MST is usually preferred because it offers a better Lagrangian convergence [2]. Indeed, the best results are obtained when using the above relaxations in a Lagrangian way [2, 4, 6].

4 Considering the reduced graph

In this section, we consider a subproblem which is not a subset of constraints, as usual, but consists in the whole ATSP itself applied to a more restrictive scope: the reduced graph of G_P. In this section, we first study structural properties

that arise from considering the reduced graph. Second, we show how to adapt such information to the Minimum Spanning Tree relaxation.

4.1 Structural properties

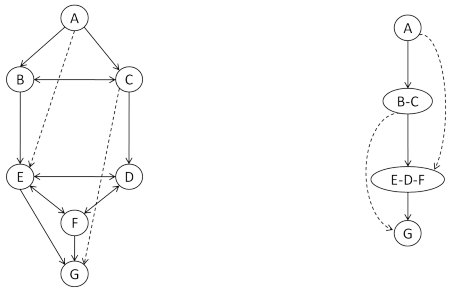
We introduce a propagator, the `Reduced Path` propagator, which makes the reduced graph a (Hamiltonian) simple path and ensures by the way that each node is reachable from s and can reach e.

Definition 1. *Reduced path guarantees that any arc in G_P that connects two SCCs, is part of a simple path which go through every SCC of G_P.*

Such a propagator has already been highlighted in [3] and comes from the two following general observations:

- Given any directed graph G, its reduced graph G_R contains at most one Hamiltonian path.
- If there exists a Hamiltonian path in G then there exists a Hamiltonian path in G_R.

It follows that any transitive arc of G_R must be pruned and that remaining arcs of G_R are mandatory (otherwise the graph becomes disconnected). We note $e_R = \texttt{sccOf}(e)$. Then, any SCC, but e_R, must have exactly one outgoing arc. An example is given in figure 1: the graph G_P contains four SCCs. Its reduced graph, G_R, has a unique Hamiltonian path $P_R = (\{A\}, \{B, C\}, \{E, D, F\}, \{G\})$. Arcs of $G_R \backslash P_R$ are infeasible so (A, E) and (C, G) must be pruned from G_P.



(a) G_P (b) G_R

Fig. 1. `Reduced Path` filtering, transitive arcs (dotted) are infeasible.

We introduce a new data structure in G_R that we call `outArcs` : for each node $x \in V_R$, $\texttt{outArcs}(x)$ is the list of arcs $\{(u, v) \in A_P |\ \texttt{sccOf}(u) = x$ and $\texttt{sccOf}(v) \neq x\}$. We can now easily draw a complete filtering algorithm for the `Reduced Path` propagator which ensures the GAC over the property that G_R must be a path in $O(n + m)$ time:

1. Data structures: Compute the SCCs of G_P (with Tarjan's algorithm [12]) and build the reduced graph $G_R = (V_R, A_R)$.
2. Taking mandatory arcs into account: $\forall (u,v) \in A_M$ such that $x = \texttt{sccOf}(u)$ and $x \neq \texttt{sccOf}(v)$, $\forall (p,q) \in \texttt{outArcs}(x) \setminus \{(u,v)\}$ remove arc (p,q).
3. Consistency checking: Make G_R a path if possible, fail otherwise.
4. for each arc $(u,v) \in A_P$ such that $x = \texttt{sccOf}(u)$ and $y = \texttt{sccOf}(v)$, $x \neq y$,
   (a) Pruning: if $(x,y) \notin A_R$, remove arc (u,v).
   (b) Enforcing: if $(x,y) \in A_R$ and (u,v) is the only arc of A_P that links x and y, enforce arc (u,v).

An incremental algorithm is provided in [6].

In general, the reduced graph provides three kinds of information: Precedences between nodes of distinct SCCs; Reachability between nodes of the graph; Cardinality sets $\forall x \in V_R \setminus \{e_R\}, |\texttt{outArcs}(x)| = 1$. Such information can be used to improve the model. For instance, one can filter on time windows (if any) through a simple graph exploration that process SCCs one after the other, in linear time. However, our main interest is to show how both the MST and the MSA relaxations of the TSP can be improved by considering the reduced graph.

4.2 The Bounding Spanning Tree relaxation

We call a Bounding Spanning Tree (BST) of G_P a minimum cost spanning tree of G_P such that, for each pair of SCCs $(a,b) \in G_R$, it has at most one edge with one extremity in a and the other in b. A BST provides a tighter bound than a MST. Indeed, since BST and MST both are spanning trees, $f(BST(G_P)) \geq f(MST(G_P))$, otherwise MST is not minimal. Such a BST can be obtained by finding a minimum spanning tree in every SCC of G_P independently and then linking them together using the cheapest arcs:

$$BST(G_P) = \bigcup_{x \in G_R} MST(G_P \bigcap \texttt{nodesOf}(x)) \\ \bigcup_{a \in V_R} min_f\{(u,v) \mid (u,v) \in \texttt{outArcs}(a)\}$$

A simpler way to compute a BST consists in computing a MST on a transformed input cost matrix, where arcs between distinct SCCs are penalized:

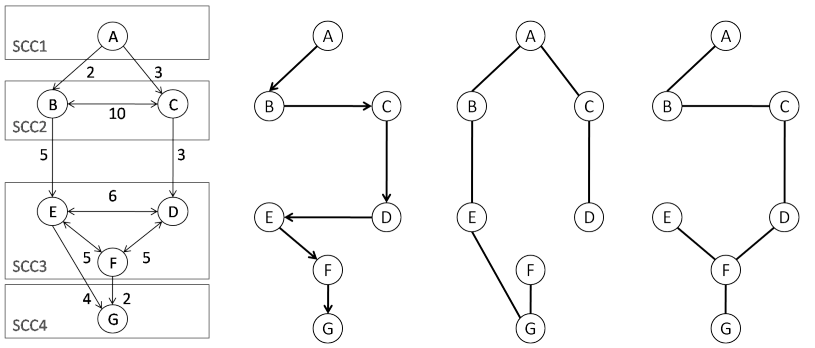
$$f'(a,b) = \begin{cases} f(a,b) \text{ If } \texttt{sccOf}(a) = \texttt{sccOf}(b) \\ f(a,b) + K \text{ Otherwise, } K \geq 0 \end{cases}, \forall (a,b) \in A_P$$

Where K is a positive offset that must then be deduced from the cost of the resulting MST: $f(BST) = f'(MST(G'_P)) - K * (|V_R| - 1)$ with $G'_P = (V, A_P, f')$. The idea is that, if arcs between SCCs are expensive enough (which is the case if K is an upper bound of the problem), then any MST will, naturally, use no more than one of them to link two SCCs. So $f(BST(G_P)) = f(MST(G'_P))$ because $MST(G'_P)$ contains exactly $(|V_R| - 1)$ arcs whose cost has been increased by K.

We will now see how to improve the `Weighted Spanning Tree` (WST) [11] constraint, leading to the `Bounding Spanning Tree` (BST) propagator. We assume that the reader is already familiar with this constraint. The BST can replace the MST of the WST constraint: the pruning rules of WST constraint will provide more inference since the bound is tighter. Actually, we can do

even better by slightly modifying the pruning rule of the WST constraint for arcs that are between two SCCs: an arc linking two SCCs can only replace (or be replaced by) another arc linking those two same SCCs. Consider any SCC $x \in V_R \backslash \{e_R\}$, there is one single tree arc $(u, v) \in \texttt{outArcs}(x)$ and then we can rephrase the pruning rule by: Any arc $(u_2, v_2) \in \texttt{outArcs}(x)$ is infeasible if $f(BST) - f(u, v) + f(u_2, v_2) > UB$, where UB is the upper bound of the objective variable.

Figure 2 illustrates this relaxation : the input directed graph, on figure 2(a), is composed of four SCCs $\{A\}, \{B, C\}, \{E, D, F\}$ and $\{G\}$. For simplicity purpose, costs are symmetric. Its minimum hamiltonian path, figure 2(b), costs 28 and we will suppose that such a value is the current upper bound of the objective variable. The MST of the graph, figure 2(c), only costs 19, which is unfortunately too low to filter any arc. Instead, the BST, figure 2(d), is much more accurate. It actually consists of the MST of each SCC, $\{\emptyset, \{(BC)\}, \{(D, F), (E, F)\}, \emptyset\}$ with respective costs $\{0, 10, 10, 0\}$, and the cheapest arcs that connect SCCs each others: $\{(A, B), (C, D), (F, G)\}$ with respective costs $\{2, 3, 2\}$. Thus, the entire BST costs 27. It is worth noticing that it enables to filter arcs (B, E) and (E, G). Indeed, (B, E) can only replace (C, D) in the relaxation, so its marginal cost is $f(BST) + f(B, E) - f(C, D) = 27 + 5 - 3 = 29$ which is strictly greater than the upper bound of the objective. The same reasoning enables to prune (E, G).



(a) Input graph (b) Optimum= 28 (c) MST bound= 19 (d) BST bound= 27

Fig. 2. A new tree relaxation, more accurate than the MST.

4.3 The Bounding Spanning Arborescence relaxation

In the same way, we can define a Bounding Spanning Arborescence (BSA). The BSA of G_P is obtained by using the above mentioned cost transformation and then computing a MSA in G'_P. It costs $f(BSA) = f'(MSA(G'_P)) - K * (|V_R| - 1)$. Again, a BSA is a spanning arborescence, so we have $f(BSA) \geq f(MSA)$. This can provide a better bound and thus, more filtering. However, the new filtering rule introduced for the BST is not valid anymore for the BSA. This is because changing the arc that link two distinct SCCs may change the whole MSA.

5 Conclusion and Perspectives

We have seen how to strengthen some existing relaxations by considering structural information of the reduced graph. In [6] we experimentally compared the Lagrangian relaxation of Held and Karp with a BST-based Lagrangian relaxation. It appeared that the BST enabled to improve some pathological cases. However, on the overall, results did not meet our expectations because of some instability during the convergence of the Lagrangian process. As future work, we think it would worth working on making such a Lagrangian relaxation more stable. We also plan to study the impact of such relaxations on more constrained problems, such as the TSPTW.

References

1. David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2006.
2. Pascal Benchimol, Willem-Jan Van Hoeve, Jean-Charles Régin, Louis-Martin Rousseau, and Michel Rueher. Improved filtering for Weighted Circuit Constraints. *Constraints*, To appear.
3. Hadrien Cambazard and Eric Bourreau. Conception d'une contrainte globale de chemin. In *Journées Nationales sur la résolution Pratique de Problèmes NP Complets, JNPC*, pages 107–120, 2004.
4. Yves Caseau and François Laburthe. Solving Small TSPs with Constraints. In *International Conference on Logic Programming, ICLP*, pages 316–330, 1997.
5. Grégoire Dooms, Yves Deville, and Pierre Dupont. CP(Graph): Introducing a Graph Computation Domain in Constraint Programming. In *Principles and Practice of Constraint Programming, CP*, volume 3709, pages 211–225, 2005.
6. Jean-Guillaume Fages and Xavier Lorca. Improving the Asymmetric TSP by Considering Graph Structure. Technical report, Ecole des Mines de Nantes 12/4/INFO, 2012.
7. Filippo Focacci, Andrea Lodi, and Michela Milano. Embedding Relaxations in Global Constraints for Solving TSP and TSPTW. *Annals of Mathematics and Artificial Intelligence*, 34(4):291–311, 2002.
8. Gilles Pesant, Michel Gendreau, Jean-Yves Potvin, and Jean-Marc Rousseau. An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows. *Transportation Science*, 32(1):12–29, 1998.
9. Jean-Charles Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *National Conference on Artificial Intelligence, AAAI*, pages 362–367, 1994.
10. Jean-Charles. Régin. Tutorial: Modeling Problems in Constraint Programming. In *Principles and Practice of Constraint Programming, CP*, 2004.
11. Jean-Charles Régin, Louis-Martin Rousseau, Michel Rueher, and Willem Jan van Hoeve. The Weighted Spanning Tree Constraint Revisited. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR*, volume 6140, pages 287–291, 2010.
12. Robert E. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

Dynamic Virtual Arc Consistency

Hiep Nguyen[1], Thomas Schiex[1], and Christian Bessiere[2]

[1] Unité de Biométrie et Intelligence Artificielle, INRA, Toulouse, France
[2] University of Montpellier, France

Abstract. Virtual Arc Consistency is a recent local consistency for processing cost function networks that exploits a simple but powerful connection between classical constraint networks and cost function networks. The algorithm enforcing virtual arc consistency iteratively solves a sequence of classical constraint networks. In this work, we show that dynamic arc consistency algorithms can be suitably injected in the virtual arc consistency iterative algorithm, providing noticeable speedups.

1 Introduction

Graphical model processing is a central problem in AI. The optimization of the combined cost of local cost functions, central in the valued CSP framework [4], captures problems such as weighted MaxSAT/CSP or Maximum Probability Explanation in probabilistic networks. Depth First Branch and Bound search has been largely used to tackle such problems. It has a reasonable space complexity but requires good lower bounds on the minimum cost to be efficient.

In the last years, increasingly better lower bounds have been designed by enforcing local consistencies on CFN. Enforcing is done using so-called *Equivalence Preserving Transformations* (EPTs, [2]) which extend usual CSP local consistency operations. EPTs move costs between cost functions while keeping the problem equivalent. They may eventually increase the cost function of empty scope (a constant) to a non naive value. This value provides a lower bound on the optimum cost which can be maintained during branch and bound search.

Virtual arc consistency (VAC), introduced in [1], relies on pre-planned applications of EPTs built from the result of enforcing classical arc consistency (AC) on a constraint network called Bool(P) which forbids combinations of values with non zero costs in the original CFN P. VAC dominates all previously defined chaotic local consistencies, and it can be approximately enforced with a low order polynomial time iterative algorithm. Maintaining VAC during tree search has effectively allowed to close two difficult instances of Radio Link Frequency Assignment instances. Each iteration of VAC incrementally modifies the network P. The next iteration therefore proceeds by enforcing classical AC on a slightly relaxed version of Bool(P). This situation, where AC is iteratively enforced on incrementally modified versions of a constraint network, has been previously considered in dynamic arc consistency algorithms for dynamic CSPs [3]. In this paper we adapt ideas from dynamic AC in the last phase of VAC. We observe that this new version frequently provides significant speedups. This may be, as far as we know, one of the first successful application of dynamic AC algorithms.

2 Background

A Cost Function Network (CFN), or weighted CSP (WCSP) is a tuple (X, D, W, m) where X is a set of n variables. Each variable $i \in X$ has a domain $D_i \in D$. For a set of variables S, we denote by $\ell(S)$ the set of tuples over S. W is a set of e cost functions. Each cost function $w_S \in W$ assigns costs to assignments of variables in S i.e. $w_S : \ell(S) \rightarrow [0..m]$ where $m \in \{1, ..., +\infty\}$. The addition and subtraction of costs are bounded operations, defined as $a \oplus b = \min(a + b, m)$, $a \ominus b = a - b$ if $a < m$ and m otherwise. The cost of a complete tuple t is the sum of costs $Val_P(t) = \bigoplus_{w_S \in W} w_S(t[S])$ where $t[S]$ is the projection of t on S. We assume the existence of a unary cost function w_i for every variable, and a nullary cost function, noted $w_\varnothing$. This constant positive cost defines a lower bound on the cost of every solution. In this paper, we restrict ourselves to binary CFNs.

Enforcing a given local consistency on a CFN P transforms it in an equivalent problem P' ($Val_P(t) = Val_{P'}(t)\ \forall t$) with a possible increase in the lower bound $w_\varnothing$ on the optimal cost. Enforcing is done by using equivalence-preserving transformations (EPTs) which shift costs between cost functions. There are three basic EPTs. $\mathsf{Project}(w_{ij}, i, a, \alpha)$ moves an amount of cost α from a binary cost function to a unary one. Conversely, $\mathsf{Extend}(i, a, w_{ij}, \alpha)$ sends an amount of cost α from a unary cost function to a binary one. Finally, $\mathsf{UnaryProject}(i, \alpha)$ projects an amount of cost α from a unary cost function to the nullary cost function $w_\varnothing$.

In a classical binary CSP, represented as a CFN with $m = 1$ (cost 1 being associated to forbidden tuples), a value (i, a) is AC w.r.t. a constraint w_{ij} iff there is a pair (a, b) that satisfies w_{ij} (is a support) and such that $b \in D_j$ (is valid). A CSP is AC if all its values are AC w.r.t. to all constraints. Enforcing AC on a CSP produces its AC closure, which is equivalent to P and is AC.

Definition 1. *Given a CFN $P = (X, D, W, m)$, the CSP Bool$(P) = (X, D, \overline{W}, 1)$ is such that $\exists \overline{w}_S \in \overline{W}$ iff $\exists w_S \in W$, $S \neq \varnothing$ and $\overline{w}_S(t) = 1 \Leftrightarrow w_S(t) \neq 0$. A CFN P is virtual arc consistent (VAC) iff the AC closure of Bool(P) is non-empty.*

If P is not VAC, there exists a sequence of EPTs which leads to an increase of $w_\varnothing$ when applied on P. VAC enforcing uses an iterative three-phases process. The first phase is an instrumented AC enforcing on the CSP Bool(P) that records every deletion in a dedicated data-structure denoted as killer. When a value (i, a) lacks a valid support on $\overline{w}_{ij}$, we set $\mathsf{killer}((\mathrm{i,a})) = j$ and we delete the value. If no domain wipe-out occurs, P is VAC and we stop. Otherwise, phase 2 identifies the subset of value deletions that are necessary to produce the wipe-out and stores them in a queue R by tracing back the propagation history defined by killer. Phase 2 also computes the maximum possible increase achievable in $w_\varnothing$, denoted λ, and the set of EPTs to apply to P in order to achieve this increase. All the amounts of cost that the EPTs will move are stored in two arrays of integers, $k(j, b)$ and $k_{ij}(j, b)$, that store the number of quantum λ that needs to be respectively projected on (j, b) and extended from (j, b) to w_{ij}. These cost moves follow a simple law of conservation. For any value (j, b) which is not a source of cost ($w_j(b) = 0$), the amount of cost that arrives in (j, b) by $\mathsf{Project}$ is exactly the amount of cost that leaves (j, b) by Extend (See [1], page 465).

Phase 3 of VAC modifies the original CFN by applying the EPTs defined by $k()$ and $k_{ij}()$ on all the deleted values that have been stored in R. A value (j,b) deleted by $\overline{w}_{ij}$ will receive a cost of $k(j,b) \times \lambda$ by Project from w_{ij}. This requires to first extend a cost $k_{ij}(i,a) \times \lambda$ from the invalid supports (i,a) to w_{ij}. The result of this phase is a new problem P', equivalent to P but with an increased lower-bound $w_\varnothing$. Ultimately, VAC iterations enforce AC on a sequence of slightly modified CSPs: Bool(P), Bool(P'), ... This motivates the use of dedicated dynamic AC algorithms (DnAC) to enforce VAC. The aim of DnAC algorithms is to maintain AC in CSP problems after each constraint addition or retraction.

Several algorithms have been proposed for DnAC. In this paper, we will use AC/DC2 [5]. AC/DC2 uses the data structure *justification*(i,a) to remember the cause of deletion for (i,a). In the initialization stage, only values which have been deleted because of the removed constraint c_{ij} are considered as candidates for restoration. In the propagating stage, a variable i having restored values can check neighboring values (j,b) for restorability only if they have been removed due to the lack of support on the constraint c_{ji} (known through $justification$). In a last stage, all restored values need to be checked again for arc consistency.

3 Dynamic VAC algorithm

We propose an improved version of VAC, called dynamic VAC (DynVAC), which uses dynamic AC to maintain AC on the successive Bool(P) instead of refiltering from scratch at each iteration as done in the standard VAC algorithm. We use an AC/DC2 version based on AC2001 instead of AC3. This also has the advantage that the *justification* data-structure of AC/DC-2 is provided for free by the killer array in VAC. DnAC algorithms are usually applied after each constraint removal or addition. In the case of VAC, at each iteration, a series of modifications of Bool(P) can occur during Phase 3. Let us denote by P^t the CFN at the beginning of iteration t. The problem P^t has cost functions w_i^t and w_{ij}^t. We observe that the global effect of all EPTs on Bool(P) in Phase 3 can be captured as a list of relaxations only, at the unary and binary levels.

Proposition 1. *Following Phase 2 of iteration t, we know that: 1) $\forall (i,a)$: $w_i^{t+1}(a) \leq w_i^t(a)$. 2) $\forall (i,a)$ and (j,b) : if $w_{ij}^{t+1}(a,b) \neq w_{ij}^t(a,b)$ then (i,a) or (j,b) is deleted in the partial AC closure of Bool(P^t).*

Corollary 1. *The EPTs applied in the phase 3 of VAC, transforming Bool(P^t) into Bool(P^{t+1}), generate only the following types of relaxations 1) values (i,a) that become authorized ($w_i^t(a) > w_i^{t+1}(a) = 0$). 2) pairs $((i,a),(j,b)$ that become authorized ($w_{ij}^t(a,b) > w_{ij}^{t+1}(a,b) = 0$).*

Therefore, the DnAC algorithm can be specialized. The restoration protocol will consist of 3 stages , as in AC/DC2 (Algorithm 1). We denote by $\overline{D}_i$ the domain of variable i in the final justified partial AC closure obtained after Phase 1.

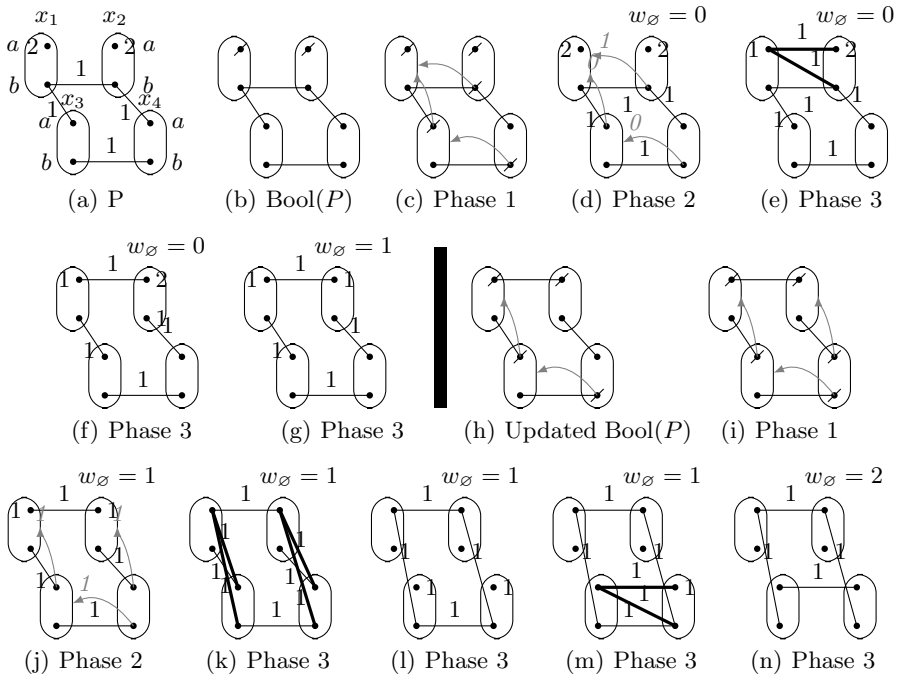
The **initialization stage** scans all the values in the queue R to identify which values should be restored. The wipe-out variable i_0 is processed separately

Algorithm 1: Update Bool(P) at iteration t

```
1  Procedure Initialization
2      foreach (j, b) ∈ R do
3          i ⟵ killer [j, b];
4          foreach a ∈ D_i − D̄_i do
5              if (w_i^t(a) > 0) ∧ (w_i^{t+1}(a) = 0) then Restore(i, a);
6              if b ∉ D̄_j ∧ w_i^{t+1}(a) = 0 ∧ w_ij^{t+1}(a, b) = 0 then Restore(j, b);
7      foreach a ∈ D_{i0} s.t. w_{i0}^t(a) > 0 ∧ w_{i0}^{t+1}(a) = 0 do Restore(i0, a);
8  Procedure Restore(i, a)
9      add a into D̄_i, restored[i] and add i into RL;
10     killer [i, a] ← nil;
11 Procedure Propagation
12     while RL ≠ ∅ do
13         i ← RL.pop();
14         foreach w_ij ∈ W do
15             foreach b ∈ D_j − D̄_j s.t. killer [j, b]= i do
16                 if ∃a ∈ restored[i] s.t. w_ij^{t+1}(a, b) = 0 then Restore(j, b);
17         restored[i] ← ∅;   Q_AC ← Q_AC ∪ {j | w_ij ∈ W}
```

(line 7). When a value (i, a) is restored, it is stored in an array *restored*$[i]$ and variable i is kept in a list RL for future propagation. The **propagation stage** propagates value restorations to direct neighbors of the variables whose domain has been extended. Each such variable i can restore a value (j, b) if it was deleted due to w_{ij} (line 15) and is now supported by a restored value in i (line 16). The **filtering stage** must eliminate the restored values (i, a) which are not arc consistent on some constraint $\overline{w}_{ij}$ and must properly set the associated killer (i, a) to j. This is precisely what is achieved by Phase 1 of VAC. Hence, we integrated this stage into phase 1 by adding the neighbor variables of variables having restored values into the revision propagation queue Q_{AC} (line 17).

Consider the binary CFN in Fig.a. Only non-zero costs and edges associated with non-zero binary costs are displayed. In Bool(P) (Fig.b), forbidden values are shown as crossed-out and edges represent forbidden pairs. The revision order in Phase 1 is (w_{13}, w_{34}, w_{12}, w_{24}). Phase 1 stops when x_2 is wiped-out after revising w_{13}, w_{34}, w_{12} (Fig.c). The gray arrows point to the variable offering no valid support and the associated italic numbers represent $k(i, a)$. In Phase 2 (Fig.d), the deletion of (x_2, b) alone is sufficient for the wipe-out. It uses the non-zero costs $w_{12}(b, b)$ and $w_1(a)$ to provide $w_\varnothing$ with a maximal amount of cost $\lambda = 1$. Applying identified EPTs, Phase 3 transforms P into an equivalent problem P^2 with $w_\varnothing = 1$ (Fig.e - Fig.g). Extended costs are shown in bold. To update Bool(P) in Fig.c, we consider $((x_1, a), (x_2, a), (x_2, b))$ since only w_{12} has been modified by EPTs in phase 3. Only (x_2, b) is restored because it has a



(a) P (b) Bool(P) (c) Phase 1 (d) Phase 2 (e) Phase 3

(f) Phase 3 (g) Phase 3 (h) Updated Bool(P) (i) Phase 1

(j) Phase 2 (k) Phase 3 (l) Phase 3 (m) Phase 3 (n) Phase 3

zero cost and a support (b, b) on w_{12}. This restoration does not lead to further restorations. The constraints of the updated Bool(P^2) are directly defined by P^2. The updated result (Fig.h) has 2 extra deleted values with associated killer. The next phase 1 starts from this updated Bool(P^2). The final problem with $w_\varnothing = 2$ is VAC.

4 Experiments

In this section, we compare the efficiency of DynVAC and VAC used as preprocessing algorithms on a large set of benchmarks from the Cost Function Library[3]. For each problem, as in [1], we enforce a limited version of VAC that stops iterating as soon as the increase in $w_\varnothing$ becomes less than $\varepsilon = 0.05$.

The following table shows the mean value of the run-time (in seconds), the lower bound (lb) and the number of iterations (iter) for enforcing VAC using either the usual static VAC algorithm or the new DynVAC variant. Each line corresponds to a different problem class covering *Size* instances. These experiments show that DynVAC is respectively 1.6, 3 and 5 times faster than VAC for the classes *celar, tagsnp, warehouse* while providing similar lower bounds on average. However, DynVAC is significantly slower than VAC (respectively 7 and 4 times) for all the maximum clique problems categories we tested: *protein_mc* and *dimacs_mc*. On these problems, we noticed that each iteration leads to the

[3] https://mulcyber.toulouse.inra.fr/scm/viewvc.php/trunk/?root=costfunctionlib

class	Size	VAC_ε			$DynVAC_\varepsilon$			$DynVAC_\varepsilon$ with heuristic		
		lb	time	iter	lb	time	iter	lb	time	iter
celar	32	6,180	3.14	382	**6,204**	**1.92**	418	5,892	1.12	319
protein_mc	10	1,016	**51**	1,022	1,016	364	1,022	1,016	56.95	1,022
tagsnp_r0.5	25	1.43×10^6	364.31	8,798	1.43×10^6	**116.57**	4,653	1.43×10^6	81.46	5,810
tagsnp_r0.8	82	1.11×10^6	4.64	155	1.11×10^6	**1.53**	120	1.11×10^6	2.54	150
dimacs_mc	65	266	**0.78**	284	266	3.65	284	266	0.96	284
planning	68	1,074	0.25	46	1,074	**0.19**	50	1,072	0.23	76
warehouse	57	7.23×10^6	341	946	$\mathbf{7.24}\times10^6$	**66**	719	7.25×10^6	114.17	790

useless restoration of many values in cascade which will again be uselessly deleted in the next iteration. In order to improve the efficiency of DynVAC we have used the variable-based revision heuristics proposed in [6]. The corresponding results are presented in the last column of the above table. They show that the heuristics allows to drastically improve the performance of DynVAC on maximum clique problems, leading to performances which are comparable to the static VAC in this highly unfavorable case.

5 Conclusion

We have proposed an incremental approach for enforcing VAC in CFNs. It combines the idea of DnAC algorithms with the iterative VAC algorithm in order to efficiently maintain arc consistency in the CSP Bool(P) during VAC enforcing. The new algorithm, DynVAC, provides a lower bound of the same quality level as the static VAC algorithm but is faster than static VAC on many problems. However, DynVAC may become slow on some specific problems like the *maximum clique* problems. Using a revision heuristic on domain sizes inside the AC instrumented algorithm allows to avoid this pathological behavior.

References

1. Cooper, M., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., Werner, T.: Soft arc consistency revisited. Artificial Intelligence 174, 449–478 (2010)
2. Cooper, M.C., Schiex, T.: Arc consistency for soft constraints. Artificial Intelligence 154(1-2), 199–227 (2004)
3. Dechter, R., Dechter, A.: Belief maintenance in dynamic constraint networks. In: Proc. of AAAI'88. pp. 37–42. St. Paul, MN (1988)
4. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: hard and easy problems. In: Proc. of the 14^{th} IJCAI. pp. 631–637. Montréal, Canada (Aug 1995)
5. Surynek, P., Barták, R.: A new algorithm for maintaining arc consistency after constraint retraction. In: Proc. Principles and Practice of Constraint Programming – CP 2004. pp. 767–771. No. 3258 in LNCS, Toronto, Canada (2004)
6. Wallace, R., Freuder, E.: Ordering heuristics for arc consistency algorithms. In: Proceedings of the Biennial Conference-Canadian Society for Computational Studies of Intelligence. pp. 163–163 (1992)

CSBT: Constrained Search-based Test Data Generation for Software

Abdelilah Sakti[1], Yann-Gaël Guéhéneuc[2], and Gilles Pesant[2]

[1]: PhD Student; [2]: Supervisor
Department of Computer and Software Engineering
École Polytechnique de Montréal, Québec, Canada
{abdelilah.sakti,yann-gael.gueheneuc,gilles.pesant}@polymtl.ca

Abstract. Automated test data generation can significantly reduce software cost, development time, and time to market. In this paper, we propose a novel combination of constraint-based testing and search-based testing, two dominant approaches to software testing: search-based testing is used to generate test data while constraints are used to guide the generation of test data candidates, to rank them, and to evolve them. Empirical results on three open-source programs show improvements in terms of branch coverage while reducing computation time.

Keywords: Search Based Testing, Constraint Based Testing, Initial Population Generator, Fitness Function, Evolution Operator.

1 Introduction

Test case generation is one of the most expensive parts of the software testing phase. Automating testing could significantly reduce software cost, development time, and time to market [2]. During the last decade, the automatic generation of structural software test data has received much attention in the literature [3, 4, 6]. Constraint-based testing (CBT) and search-based testing (SBT) approaches have become the dominant approaches in this research area because they achieve high code coverage. Yet, these approaches suffer some problems.

The advantages of CBT are the precision of its generated test data and its ability to prove that some paths are unreachable. Its main disadvantage is its inability to handle dynamic aspects of unit under tests (UUTs): dynamic structures and native function calls. Thus, CBT cannot always generate test data due to source code complexity and unavailability. In contrast, SBT can handle any type of UUTs but is sensitive to the search-space size, the diversity of the initial population, the effectiveness of its evolution operators, and the quality of its fitness function. SBT cannot handle certain aspects of UUTs: nested structures, flags [5] while these can be handled by CBT.

In this paper , we propose a hybrid approach, *CSBT*, that generates test data by combining CBT and SBT. CSBT models a relaxed version of a UUT as a constraint satisfaction problem (CSP). Based on this model and the test target, a constrained population generator (CPG) generates an initial population. Then, an evolutionary algorithm uses this population, a constrained fitness function (CFF), and a constrained evolution operator (CEO) to generate test data leading to cover the test target.

```
intStr(int X,int Y,
    String S1,String S2) {
  int y= X<<Y;
  int x=y+X/Y;
  String s=S1+S2;
  if((s.equals("OK")
    && (x>0)
    && s.length()>x)
      return 1; // Target
  return 0;
}
```

Fig. 1: intStr function —

```
intStr(int X,int Y)
                    {
  int R1,R2; int y= R1;
  int x=y+X/Y;

  if(
    (x>0)
    && R2>x)
      return 1; // Target
  return 0;
}
```

Fig. 2: Relaxed version for an integer solver

```
intStr(
    String S1,String S2) {


  String s=S1+S2;
  if(s.equals("OK"))


      return 1; // Target
  return 0;
}
```

Fig. 3: Relaxed version for a string solver

We report on the comparison of a CBT, a SBT, and our CSBT approach and show that the latter outperforms the others on our benchmark UUTs. The empirical results show that CSBT reduces the effort needed to reach a given test target and that it achieves higher branch coverage than the test data generated by each approach alone.

The remainder of the paper is organized as follows: Section 2 briefly describes our testing approach. Section 3 presents an empirical study. Section 4 summarises related work. Section 5 concludes with some future work.

2 CSBT: Constrained Search-based Testing

The proposed approach can be used with any combination between a static CBT and any SBT that is based on either population-based or multi-start trajectory-based metaheuristic.

We assume that: (1) using a diversified initial population that partly satisfies a UUT's predicates leading to the test target can reduce significantly the effort required to reach this test target; (2) test candidates that meet complex branches are more promising than those that meet less complex branches and, therefore, a fitness function based on branch complexity can provide efficient guidance; (3) test candidates that satisfy some test target constraints are "closer" to the test target than those that are randomly selected. Therefore an evolution operator that evolves population according to a UUT's CSP model can be more effective than an ordinary one.

2.1 Unit Under Test Relaxation

To avoid the CBT limitations, we introduce a preliminary phase called UUT relaxation. We propose to apply CBT on a relaxed version of the UUT, which contains only expressions supported by the constraint solver used. We obtain this version by applying the following rules:

– We relax any unsupported expression (function call, operator) by replacing it with a new variable. Figure 2 shows a relaxed version of `intStr`, which is shown in Figure 1, for an integer solver. We suppose that the solver cannot handle the shift operator, so we replace the expression with this operator by a new, uninitialised variable `R1`, which can take any integer value.

– We ignore any statement over unsupported data types. In Figure 2, the relaxed version of `intStr` ignores the statement at line 5 (`String s=S1+S2;`) and the a part of the conditional statement at line 6 (`s.equals("OK")`) because these two expressions are over strings.
– We create a new relaxed version for each data type that needs a different solver. The function `intStr` takes integer and string types as inputs. If we have a solver for strings, then we generate another relaxed version over strings. Figure 3 shows a relaxed version of `intStr` for a string solver.
– For loops, CBT models a limited number of iterations: `kpath` (equals to 1, 2, or 3). If the number of iterations is greater than `kpath`, then the value of an assigned variable inside a loop is unknown after this loop. We relax any variable assigned inside a loop just after the loop, i.e., an assignment statement is inserted after the loop for every variable defined inside the loop: a variable is assigned a new uninitialized variable.

We use an adapted version of our CBT approach [8] that we call *relaxed CBT* (RCBT). RCBT takes as input a relaxed version of the UUT and a test target to meet. Then, it produces, if feasible, a set of pseudo test inputs that covers a test target in the relaxed version.

2.2 Collaboration between RCBT and SBT

The collaboration between RCBT and SBT is as follows. RCBT generates a pseudo test data and then SBT uses this pseudo test data to generate an actual test data. One of our contributions is to define the data exchanged and connection points between RCBT and SBT. SBT needs new test input candidates at three different points: (1) during the generation of its initial population; (2) when it restarts, i.e., when it reaches the attempt limit of evolving; (3) during its evolving procedure. For the first and the second points, RCBT can generate the whole or parts of the population. If RCBT is unable to generate the required number of candidates, we call the random generation procedure to complete the population. For the third point, RCBT can offer a branch complexity measure for a fitness function and help in the population evolution process by discarding candidates that break any relaxed models and by allowing only candidates that satisfy all relaxed models.

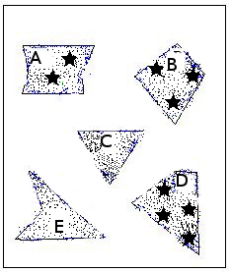


Fig. 4: Inputs search space.

Constrained Population Generator. For each relaxed version, CPG generates a set of pseudo test data. Then, the SBT initial population is generated from all possible combinations of these pseudo test inputs, i.e., we combine (Cartesian product) every solution of one relaxed version with every solution of other relaxed versions.

CPG can be seen as a domain reduction phase. Figure 4 illustrates the inputs search space of a program `P`, the parts `A`, `B`, `C`, `D`, and `E` are the RCBT solutions space of all relaxed versions of `P`, which we call *pseudo test data*, while starts

are true, desired test inputs. This example shows that a random test candidate, from the white space, is likely to be too far from a true, desired test data when compared to some pseudo test data. The parts `C` and `E` do not contain any test data, so a pseudo test data from these two parts may also be far from a true, desired test data. A population that takes its candidates from different parts is likely to be near one of the true, desired test data; we call it a *diversified population*. We can offer an acceptable level of diversity by generating a pseudo test data for every branch consistent with a test target and a high-level of diversity by generating a pseudo test data for every path that leads to the test target.

Constrained Fitness Function (CFF). To generate test data, SBT uses a meta-heuristic guided by a fitness function. The measures *branch-distance* and *approach-level* are largely used for computing the fitness function value [5]. These measures are respectively the number of branches leading to the test target that were not executed by the test candidate and the minimum required distance to satisfy the branch condition where the execution of the test candidate has left the path's target, e.g., if the branch $x = y$ must be true, then the branch-distance can be $|x - y|$. A fitness function based on these two measures does not take into account neither condition complexity nor branch-distance of non-executed branches. A CFF that uses these two information could lead to a more effective and efficient search. We hypothesise that the complexity level of a branch condition is a key information to measure the relevance of a test candidate. Therefore, we propose to analyse, statically, the non-executed branches and to replace the approach-level measure with complexity measure.

To represent the level of complexity to satisfy a branch constraint, we introduce a new measure called *complexity level*, which depends on two parameters: the solution count of the constraint and the size of the search space. The complexity level of a constraint is defined by its tightness ($solution\_count/search\_space\_size$) [10]. Using the approach proposed in [7], we can approximate the solution count for program statements (constraint).

Constrained Evolution Operator. CEO can be a selection, crossover, or mutation operator or a neighbourhood generator. In this work, we consider that CEO is a mutation operator. With a predefined likelihood, the genetic algorithm (CSBT) calls CEO by sending the methods under test, the current test target, current input values, and the input value to change. CEO uses this information to choose the adequate CSP model (over integers, over string) and to fix the test target and CSP variables except those required to change. The model is then solved and a new input values are generated. If the CSP is infeasible, then all new input values are generated arbitrary.

3 Empirical Study

The *goal* of our study is to compare CSBT against CBT and SBT and identify the best technique among CPG, CEO, and a combination thereof, CPEO. The *quality focus* is the performance, in term of code coverage and runtime, of the

CPG technique, the CEO technique, CPEO, RCBT alone, and an open source SBT tool eToc [9] to generate test inputs that cover all branches. The *context* includes three UUTs: Integer, BitSet, and ArithmeticUtils; Java classes from the standard library and Apache Commons project[1]. The number of all branches is 285: 38 from Integer, 145 from BitSet, and 102 from ArithmeticUtils.

3.1 Analysis

Figure 5 shows the percentage of branch coverage achieved for the UUTs Integer, BitSet, and ArithmeticUtils: the CPG technique is the most effective and eToc is better than the proposed techniques during the first twenty seconds . After this lapse of time, CPG outperforms all techniques in terms of runtime and branch coverage. Also, CEO and CPEO perform better than eToc though not as well as CPG because of the frequency of solver calls: CEO calls the solver for every mutation, whereas CPG calls the solver only to generate the initial population. Therefore, CSBT boosts SBT implemented in eToc.

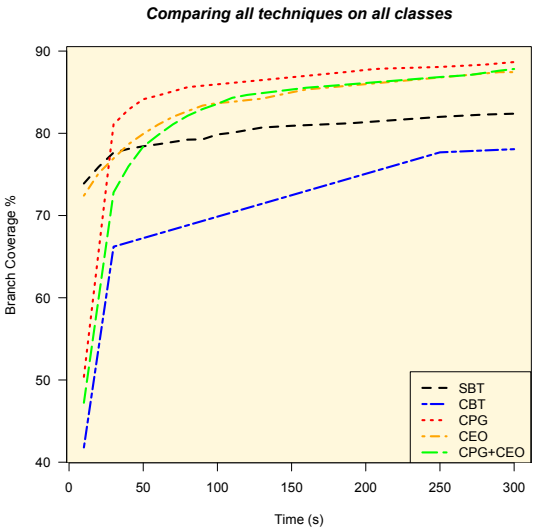


Fig. 5: Comparing all techniques

CPG enhances eToc by an average of 6.88% on all classes: 3.60% on Integer, 5.65% on BitSet, and 11.37% on ArithmeticUtils. These values do not take into account the branches proved unreachable: just on ArithmeticUtils, CPG has proved 4 infeasible branches. According to [3], the branches not covered by eToc are difficult to cover. Therefore, an additional percentage of branch coverage ranging between 3.6% and 11.37% is a good performance for CSBT.

4 Related Work

The literature describes several approaches to use CBT or combine it with SBT to generate test data, but they apply CBT to a complete version of the UUTs [1, 3, 4] and are limited by the size and the complexity of the UUTs and by the undecidability of the test generation problem (reachability problem).

Our approach differs from previous work because it uses CBT at different points of the execution of the SBT to improve this latter and provides a general framework for combining SBT and CBT: using a relaxed version of UUTs makes CSBT flexible and enlarges its applicability to different kinds of UUTs.

[1] `http://commons.apache.org`

5 Conclusion

We presented CSBT, a novel combination of SBT and CBT to generate test data. CSBT uses a relaxed version of a UUT and of CBT solutions (pseudo test data) as test candidates passed on to SBT.

We identified three main points where CBT can be useful for SBT. For each point, we proposed a new combination technique: a CPG that uses CBT to generate test candidates; a CFF that uses CBT to guide SBT search; and a CEO that uses CBT to evolve test candidates. Preliminary empirical results showed that CPG outperforms the other techniques in terms of runtime and branch coverage. It is able to reach 89.5% branch coverage in less than 40s. Also, CEO and CPEO perform better than SBT in terms of branch coverage. These results are promising but we will perform more experiments using different kinds of solvers (string, floating point) to confirm that the proposed techniques are advantageous.

Future work includes extending CSBT by exploring new combination techniques and by implementing and comparing CFF to previous work, enhancing CEO, and using several solvers.

References

1. Gotlieb, A.: Euclide: A constraint-based testing framework for critical c programs. In: ICST. pp. 151–160 (2009)
2. Ince, D.C.: The automatic generation of test data. The Computer Journal 30(1), 63–69 (1987)
3. Inkumsah, K., Xie, T.: Evacon: A framework for integrating evolutionary and concolic testing for object-oriented programs. In: Proc. 22nd IEEE/ACM ASE. pp. 425–428 (November 2007)
4. Malburg, J., Fraser, G.: Combining search-based and constraint-based testing. In: Proceedings of the 2011 International Symposium on Software Testing and Analysis. ISSTA '11, ACM, New York, NY, USA (2011)
5. McMinn, P.: Search-based software test data generation: a survey. Software Testing Verification & Reliability 14, 105–156 (2004)
6. M.Harman, Mansouri, S.A., Zhang, Y.: Search based software engineering: A comprehensive analysis and review of trends techniques and applications (2009)
7. Pesant, G.: Counting solutions of csps: a structural approach. In: Proceedings of the 19th international joint conference on Artificial intelligence. pp. 260–265. IJCAI'05, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)
8. Sakti, A., Guéhéneuc, Y.G., Pesant, G.: Cp-sst : approche basée sur la programmation par contraintes pour le test structurel du logiciel. In: Septitièmes Journées Francophones de Programmation par Contraintes (JFPC). pp. 289–298 (June 2011)
9. Tonella, P.: Evolutionary testing of classes. SIGSOFT Softw. Eng. Notes 29(4), 119–128 (Jul 2004)
10. Zanarini, A., Pesant, G.: Solution counting algorithms for constraint-centered search heuristics. Constraints 14, 392–413 (2009)

A generic framework for solving CSPs integrating decomposition methods

L. Blet[1,3], S. N. Ndiaye[1,2], and C. Solnon[1,3]

[1] Université de Lyon - LIRIS
[2] Université Lyon 1, LIRIS, UMR5205, F-69622 France
[3] INSA-Lyon, LIRIS, UMR5205, F-69621, France
{loic.blet,samba-ndojh.ndiaye,christine.solnon}@liris.cnrs.fr

Abstract. Many real-world constraint satisfaction problems are structured, i.e., constraints are not uniformly distributed among the set of variables. This structure may be used to improve the solution process of these problems. In particular, backtracking with tree decomposition (BTD) exploits the structure to define variable ordering heuristics and to learn structural goods and nogoods which are used to avoid redundant explorations. BTD is based on a chronological backtracking search. Our goal in this paper is to investigate the interest of exploiting structure when using other approaches for exploring the search space: other complete search approaches, such as conflict directed backjumping (CBJ), but also incomplete approaches, such as decision repair (DR). To this aim, we describe a generic framework for solving CSPs, which is an extension of the framework proposed by Pralet and Verfaillie in [PV04]. Using our new generic framework, we reformulate some existing search procedures, including BTD. We also describe new search procedures which combine structural (no)goods with CBJ and DR. This generic framework allows us to experimentally evaluate the interest of exploiting the structure for different kinds of search procedures.

Keywords: Constraint satisfaction, Constraint graph, Tree-decomposition, Complete search, Local search

1 Introduction

Many real-world constraint satisfaction problems (CSPs) are structured, i.e., constraints are not uniformly distributed among variables. This structure may be used to improve the solution process of these problems. In particular, BTD [JT03] exploits the structure to define variable ordering heuristics and to collect structural (no)goods which are used to avoid redundant explorations. Experimental results have brought to the fore that these (no)goods can significantly speed up the solution process of structured instances.

BTD is based on a chronological backtracking (CB) search: it exhaustively explores the search tree by performing a depth first search until either a solution is found or inconsistency is proven. Our goal in this paper is to investigate the interest of exploiting structure when using other approaches for exploring the search space: other complete search approaches, such as conflict directed backjumping (CBJ) [Pro95], but also incomplete approaches, such as decision repair (DR) [JL02]. To this aim, we describe a

generic framework for solving CSPs, which is an extension of the framework proposed by Pralet and Verfaillie in [PV04]: the framework of Pralet and Verfaillie may be instantiated in different search procedures (such as, CB, CBJ, or DR); we extend it by adding structural (no)goods to it. Using our new generic framework, we reformulate some existing search procedures, including BTD. We also describe new search procedures which combine structural (no) goods with CBJ and DR. This generic framework allows us to experimentally evaluate the interest of exploiting the structure for different kinds of search procedures.

Section 2 introduces tree decompositions and structural (no)goods, as well as the idea of the algorithm BTD. Section 3 describes our generic framework. Section 4 presents our first experimental results. Section 5 discusses further work.

2 Backtracking with Tree Decomposition

A *tree decomposition* [RS86] of a graph $G = (X, E)$ is a tree $T = (V, F)$ so that:

- each vertex of $v_i \in V$ is called a cluster and is a set of vertices of G, i.e. $v_i \subseteq X$;
- each vertex of G belongs to at least one cluster of T, i.e. $\forall x_i \in X$, $\exists v_j \in V$, $x_i \in v_j$;
- for each edge of G, there exists one cluster of T which contains both endpoints of e, i.e. $\forall (x_i, x_j) \in E$, $\exists v_k \in V$, $\{x_i, x_j\} \subseteq v_k$;
- for each vertex x_i of G, the set of clusters of T which contains x_i (i.e. $\{v_j \in V$, $x_i \in v_j\}$) induces a connected subgraph of T.

Given a tree decomposition T and a cluster c_i, we note $root_T$ the root cluster of T, $father_T(c_i)$ the cluster which is the father of c_i in T (with $c_i \neq root_T$), and $ancestors_T(c_i)$ the set of clusters which are ancestors of c_i in T. Figure 1 displays a CSP, its constraint graph and a tree decomposition for this graph.

The *width* of a decomposition T is the size of its largest cluster minus one. For example, the width of the tree in Fig. 1(c) is 2. A tree decomposition is optimal if its width is the smallest among all possible decompositions of G.

In BTD [JT03], the tree decomposition T of the constraint graph is used to define a variable ordering heuristic: all variables in a same cluster are assigned before assigning variables of its children. (No)goods are used to deduce that a subproblem rooted in a given cluster c_i is already solved (or already proven inconsistent) with respect to a current assignment A and a set of (no)goods $(N)G$. The subproblem rooted in a cluster c_i is the CSP restricted to variables of c_i and of all clusters c_j such that $c_i \in ancestors_T(c_j)$. This subproblem is already solved (resp. proven inconsistent) with respect to a current assignment A and a set of goods G (resp. nogoods NG) if there exists a good $g \in G$ such that $g = A_{\downarrow c_i \cap father_T(c_i)}$ (resp. a nogood $ng \in NG$ such that $ng = A_{\downarrow c_i \cap father_T(c_i)}$).

For example, in Fig. 1, the subproblem rooted in $\{B, C, F\}$ is the CSP restricted to $\{B, C, F, G, H, I\}$. This subproblem is solved with respect to the assignment $\{B = 1, C = 2, F = 3\}$ and the set of goods $\{\{B = 1\}_{\{B,G,H\}}, \{F = 3\}_{\{F,I\}}\}$, whereas it is inconsistent with respect to the assignment $\{B = 2, C = 3, F = 1\}$ and the set of nogood $\{F = 1\}_{\{F,I\}}$.

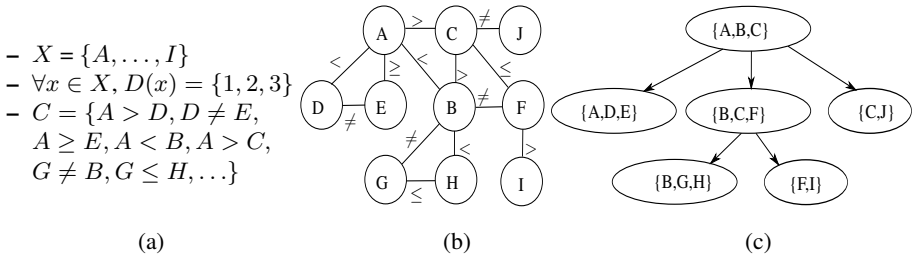


Fig. 1. (a) A CSP $P = (X, D, C)$. (b) Its constraint graph G_P. (c) A tree decomposition of G_P.

The initial problem, which is rooted in $root_T$, is solved with respect to a current assignment A and a set of goods G if $root_T \subseteq var(A)$ and every subproblem rooted in a child of $root_T$ is solved with respect to A and G.

3 A Generic Search Framework

Pralet and Verfaillie [PV04] have proposed a generic framework for defining a search. This search is parametrised by (i) filtering procedures which specify how domains are filtered after each step of the search, (ii) an assignment extension procedure, which specifies how to extend the current assignment, and (iii) a repair procedure which defines how to unassign some variables in the current assignment. This generic framework may be instantiated in various existing search procedures such as CB, CBJ or DR.

In this section, we extend the generic framework of [PV04] in order to allow its instantiation to structural approaches. To this aim, we introduce three sets:

- a set E of explanations which are recorded when filtering domains and used to restore domains when unassigning variables, to identify conflicts when backjumping, or to prove inconsistency ;
- a set G of structural goods which may be recorded and used when extending assignments to avoid redundant explorations;
- a set NG of nogoods which may be recorded and used when unassigning variables to avoid redundant explorations.

Note that explanations may be removed from E (once they are no longer relevant) whereas (no)goods are always relevant during the whole search, so that they are never removed from $(N)G$[1].

This generic framework is described in Algo.1 and is parametrised by 4 functions:

- *filter(P,A,E,NG)* ensures some local consistency (such as FC or AC) with respect to the current assignment A and returns false if some inconsistency has been detected; true otherwise.

[1] We may limit the size of G and NG. In this case we may have to remove (no)goods from $(N)G$ when the size limit is reached.

Algorithm 1: Generic Framework

Input: a CSP P and an initial assignment A
Output: Returns *true* if a solution is found; *false* if inconsistency is proven; ? otherwise

```
1  Let E, G and NG respectively be empty sets of explanations, goods and nogoods
2  repeat
3      if filter(P, A, E, NG) then
4          solved ← extend(P, A, G)
5          if solved then return true
6      else
7          inconsistent ← unassign(P, A, E, NG)
8          if inconsistent then return false
9  until Stop() ;
10 return ?
```

– *unassign(P,A,E,NG)* returns true if no more variable can be unassigned (and thus inconsistency has been proven); and false otherwise. In this latter case, it unassigns some assigned variables.
– *extend(P,A,G)* returns true if A is a solution and false otherwise. In this latter case, it extends A by adding a new variable/value couple to it.
– *stop()* returns true if a stopping criterion has been met. We consider only one instantiation, which returns true when some given CPU time limit has been reached.

3.1 algorithms

We can now combine instantiations of *extend*, *unassign* and *filter* to obtain different search strategies. Note that when using the tree decomposition you have to do a depth first search in the tree of clusters, as in BTD.

Some instantiation captures well-known search strategies such as:

– the classical Chronological Backtracking;
– Conflict-directed BackJumping [Pro95];
– Decision-Repair as described in [PV04];
– Backtracking with Tree Decomposition [JT03].

Some of these instantiations correspond to new search strategies. In particular, we define two new search strategies which exploit tree decompositions:

– CBJ-TD performs Conflict-directed BackJumping with Tree Decomposition;
– DR-TD performs Decision-Repair with Tree Decomposition.

Complexity: CB, CBJ, BTD and CBJ-TD are complete approaches. The time complexities of CB and CBJ are exponential in the number of variables, whereas the time complexities of BTD and CBJ-TD are exponential in the width of the tree decomposition (i.e. the size of the biggest cluster in the decomposition).

DR and DR-TD are incomplete approaches. Even if they may prove some inconsistencies, there is no guarantee for the execution to end. Thus their complexities are bounded by the function $stop()$.

4 First Results

We considered all the binary instances of the benchmark available at www.cril. univ-artois.fr/ lecoutre/benchmarks.html, which contains instances of the 2008 CSP competition as well as other new instances. By lack of space, we decide to not report here the tables of results but to discuss those. We compute a tree-decomposition for each instance by using the graph triangulation algorithm Minimum Fill-in which is known to be among the best methods for this purpose. The quality of this decomposition is considered good enough if its width is at most equal to half of the number of variables in the CSP (beyond, using tree-decomposition method does not derive enough benefit).

First, we observe that most instances of the benchmark are not structured. Only 5% of the binary instances fulfilled the condition: unsurprisingly, they are real-world instances (the REAL class in the benchmark) and graph coloring ones. We run the methods defined in the previous section on the instances using forward checking as filtering technique. Using tree-decomposition improves significantly the results on coloring graph instances. It allows to solve within 1 second 4 instances otherwise unsolved within 1 hour by CB and DR. CBJ-TD succeeds to solve within 3 minutes 1 instance unsolved by CBJ within 1 hour. But, tree-decomposition based method does not behave as well on instances in the REAL class. Except a quarter of instances, using tree-decomposition surprisingly degrades the performances. We try to analyze more precisely the features of the different problems and we observe that graph coloring instances are often very hard meaning that it is necessary to explore large parts of the search space to solve them leading to many redundancies. Whereas, the instances in the REAL class, despite their great size, are often easily solved with a good variable ordering (within 1 second by CB). Since, the major effect of exploiting the structure of a problem lies in reducing the number of redundancies in the resolution, it seems obvious that it performs better on hard instances. Actually, the solving is based on a variable ordering designed to learn a maximum number of (no)goods. But, the time elapsed during this learning phase, is balanced by avoiding a lot of redundancies thereafter on hard instances. This is not the case on easy ones where a more straightforward variable ordering is sufficient to solve the problem with a limited number of redundancies. Thereby, a crucial trade-off between the exploiting of the structure and the freedom given to variable heuristic must be accomplished.

Moreover, it is clear that the exploitation of the structure has more impact on CB than on CBJ. Indeed, CBJ, with its mechanism of backjumping, already integrates a clever tool to reduce redundancies. Therefore, even though CBJ-TD improves results, the gap is not so important. Finally, we notice that DR does not fare well when compared to basic algorithms such as CB and CBJ. It seems to us that this is due to the persevere heuristic [PV04] which drastically reduces possible choices when unassigning variables. Chances are there will be few variables causing the failure in the current cluster and among those it is unlikely that there will be two or more variables with the least occurrences in the explanations. This might even lead to some simple infinite loops between two or three variables.

We also ran experiments on randomly generated structured instances, with the method described in [JNT06]. We draw the same conclusions than with the CSP competition benchmark, but we also noted two interesting facts. First, DR fares as well as the best

algorithms (BTD, CBJ, CBJ-TD) on consistent instances with very dense clusters. We attribute this to the freedom given to DR, as it will be able to make interesting choices when backtracking, as there should be many variables to choose from, thus leading quickly to a solution. Second, CBJ performs worse on inconsistent instances with very dense clusters than on any other type of tested instances. This is most likely due to the fact that it will mimic the behaviour of CB, as every variable in the cluster is likely to cause a failure. Interestingly, CBJ-TD does not have this drawback.

5 Conclusion and Further work

We have described a generic framework for solving CSPs, which is an extension of the framework proposed by Pralet and Verfaillie in [PV04] in order to allow us to exploit the problem structure during the search and evaluate its interest when using different approaches for exploring the search space: chronological backtracking, Conflict directed BackJumping and Decision Repair. We have experimentally compared these different methods on structured binary instances, and we have found that exploiting the structure helps a lot for reducing redundancies when solving of an instance if this instance is "hard" enough. However, it can degrade the results for CSP instances "easily" solved by a clever traversal of the search space thanks to a good variable ordering heuristic.

The DR instantiation follows the persevere heuristic proposed in [PV04]. Experimental results have shown that this heuristic may lead to an over-intensification of the search process. It could be improved by adding some diversification mechanisms such as, for example, weighting the choice of the variable to unassign with the number of their occurrences in the explanations. This would prevent simple infinite loops.

There is still room for improvements in the way algorithms exploit the tree decomposition. In particular, we could let the variable heuristic decide which cluster to go first and give it more freedom during the search. It could go to the most difficult part first, thus recording less (no)goods in the solving process. It would also be more likely to find a solution or to prove inconsistency faster.

We plan to define an instantiation of the *filter* function which maintains Arc Consistency, and to extend our framework to non binary CSPs.

References

[JL02] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artif. Intell.*, 139(1):21–45, 2002.

[JNT06] P. Jégou, S. N. Ndiaye, and C. Terrioux. Strategies and Heuristics for Exploiting Tree-decompositions of Constraint Networks. In *Inference methods based on graphical structures of knowledge (WIGSK'06), ECAI workshop*, pages 13–18, 2006.

[JT03] Philippe Jégou and Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artif. Intell.*, 146:43–75, May 2003.

[Pro95] Patrick Prosser. Forward checking with backmarking. In *Constraint Processing, Selected Papers*, pages 185–204, London, UK, 1995. Springer-Verlag.

[PV04] Cédric Pralet and Gérard Verfaillie. Travelling in the world of local searches in the space of partial assignments. In *CPAIOR*, pages 240–255, 2004.

[RS86] N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of treewidth. *Algorithms*, 7:309–322, 1986.

Identification of Effective Configuration in Constraint Solver Synthesis

Arūnas Prokopas
ap637@st-andrews.ac.uk

School of Computer Science, University of St Andrews, UK

Abstract Constraint solvers are complex pieces of software with many interconnecting components. These solvers generally have to make tradeoffs in their architecture due to their monolithic design. The Dominion constraint solver synthesizer, however, generates a bespoke solver tailored to the features of a given problem. This paper briefly covers the initial approach to identifying the effective configuration for these solvers.

1 Introduction

Current state-of-the-art constraint solvers, such as Choco [1], Eclipse [2], Gecode [3], or Minion [4] are monolithic: large, complex pieces of software designed to perform well on a broad range of input models. While these solvers allow some tuning to optimize them for specific problem models, for example by selecting the search strategy or adjusting the level of consistency, these optimizations are limited and generally can only be done by experts.

Due to the monolithic architecture of the solvers and complexity of constraint problems, there is no guarantee that any given solver will perform well when encountering new problems. Additionally, some components that perform extremely well in individual cases, but incurs overhead in other problems, such as constraint learning [5–7], are generally avoided to protect the average-case performance.

The Dominion [8] system attempts to address these issues by using a different approach. First, it analyses the problem model and produces a constraint solver specification, which contains the information about all the functionality the solver must have to solve the given problem model. This information is then used to synthesize a specified solver from the library of available components.

Since any of the synthesized solvers can only contain a fraction of the available components, this approach allows the library to contain the components that are only beneficial in specific cases without losing performance in solvers that do not require them. Furthermore, the synthesized solvers can be fine tuned to individual problems without making compromises along the way, which can result in very significant performance improvements in some cases.

2 Dominion Structure

To generate the specialized solvers, the Dominion synthesizer initially analyses the input problem model and converts it into an internal component. Like all other Dominion components, this new component can be divided into two independent parts: architectural level representation (Grasp [9]) and actual code (C++). The architectural representation describes what functions can be performed by the component (i.e. interfaces that it provides) and what functionality it needs from other components (i.e. sub-component lists and additional constraints).

At this point, the generator builds a component tree at architecture level, in such a way that all requirements of every component are satisfied. As mentioned before, these components can impose some constraints on each other (with some exceptions non-binary variable factories cannot work with binary propagators etc.), turning the whole component selection process into a constraint problem of its own. Furthermore, the component tree does not have a static shape, because even two components that perform the same function may have a different number of sub-components and additional constraints. This prevents the use of heuristics that can generally be applied to similar configurations problems.

3 Component Testing

The most straightforward approach to testing all of the components is generating all of the possible solvers and running them. While this can only be done with simple problems (N Queens is used as an example), this can still provide valuable insights about the various solver components. These solvers are generated by using a depth first search based algorithms that constructs the solvers by evaluating the components in the appropriate component pools and attempting to attach those components to the solver.

All of the following data is from initial testing performed on a single 3.3Ghz core on using the same version of a solver synthesizer. The results shows the time it took the synthesized solver to solve a supplied problem, not including the generation time (it takes approximately 1/10,000th of a second to generate a solver using the depth first search algorithm) and solver compile time, which takes 10 seconds on average (depending on the constraint problem) and is not affected by the choice of components.

Figure 1 demonstrates the run times of all possible solvers for N queen problem class, by generating all solutions for 3 different n queens problem instances (8, 9 and 10 queens). Since the solvers are generating systematically by changing a minimal amount of components between the solvers, in addition to showing how each of the synthesized solvers perform individually, the shape of the graph reveals additional information about the solver components.

The clear divisions implies that some of the choices are clearly more important (for example the big slow down in performance at solver 634 can be attributed to choosing the worse alldifferent propagator), while the reoccurring

patterns demonstrate that some components are independent and do not influence the performance of each other.

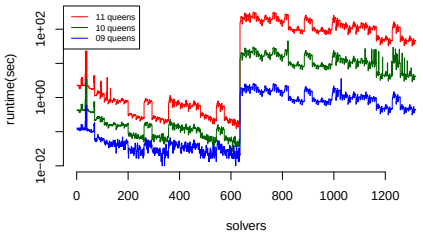


Figure 1: 11/10/9 Queens (all solutions)

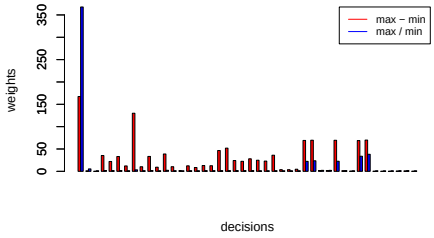


Figure 2: Choice importance

This data can then be used to rank the importance of the individual decision points during solver synthesis. To achieve this, at each decision point the solvers are divided into subsets based on the choice that is made at that decision point and mean performance of those subsets is calculated. These means can then be used to determine how well these components perform.

For example in the N queens problem one of the decision points is choosing the alldifferent propagator out of 4 different components. If we split the solvers into subsets, based on which of those propagators are chosen, we can see that the means for them are 1.1, 0.5, 167.9 and 96.8 seconds (for 11 queens, all solutions), putting two of the propagators well ahead of the others.

Furthermore, the differences and ratios between the best and the worst performing components can be used to rank the decision points themselves. It should be noted that they have to be used together, because differences, when used alone, tend to undervaluate the very important independent decisions, while ratios overinflates the value of very small components. In the case of N queens, when we compare all of the decision points in this way (figure 2), we can clearly see that two components stand out. These components are the propagator for the alldifferent constraint and variable factory that determines internal variable representation. If we only examine only the performance difference, the value seems very similar, but after we evaluate ratios as well, we can tell that the propagator is much more important.

While this approach provides a large amount of information, as mentioned before, it can only be used on simple problems that do not have many components. As the number of possible solvers grows exponentially with the number of components (for BIBD constraint problem, the number of possible solvers exceeds 4.3 million etc.) and testing all solvers is impossible.

4 Expanding to Larger Problems

During the initial testing, another observation was made: the components that appear near the top of the component tree, frequently have a significantly bigger impact on the performance of the solver. This was not unexpected as the bottom components generally contain support functions, needed by higher level components. If no additional heuristics are used, this makes the depth first search based algorithm undesirable.

To address this issue, another algorithm, based on iterative deepening was implemented. This algorithm works similarly to the depth first search based algorithm, however it maintains a depth threshold, and only tries out all component combinations above that threshold (the first available component for all the choices below the threshold is chosen to complete the solver). When all variations are tested at the current depth, the depth threshold is incremented and the algorithm restarts.

This approach can be up to 100 times slower, since it repeats a large portion of work and need to check every solver against previously generated ones. Despite this fact it still puts the generation time in the 1/100th of a second range, which is insignificant when compared to the time of compiling and testing the solver. This approach also completely disregards the bottom components, but as figure 3 demonstrates, at the same time it produces much better representation of solver variety if all solvers at the initial depth can be tested. This makes the iterative deepening based algorithm much more desirable when only small number of tests can be performed.

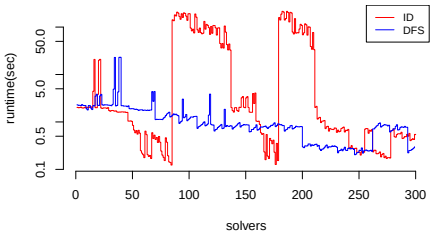


Figure 3: 11 Queens, DFS vs ID

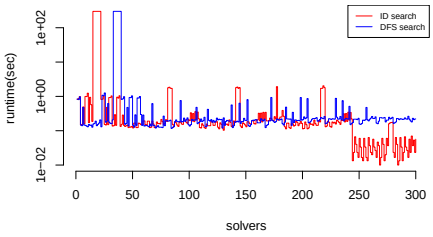


Figure 4: Golomb Ruler, DFS vs ID

While this approach can produce better variety than the depth first search even after testing a small fraction of solver as demonstrated in figure 4 (300/828 solvers at the initial depth), the improvement is not very significant, unless all solvers up to a desired depth must be tested, which may be unreasonable for larger problems. In fact, even the initial depth contains up to 10% of the total solvers, which can be equal to hundreds of solvers even for simple problems (for BIBDs it is approximately 5%, which is still equal to over 200,000 solvers etc).

5 Selective Sampling

Compiling and testing all of the possible solvers is generally impossible. On the other hand, generation of individual solvers in most cases is relatively simple. Because of this, it is possible to generate and count all of the possible solvers for most of the problem classes. As the solvers are generated systematically, this information can then be used to sample the solvers at regular intervals (as opposed to testing first solvers that are generated) to get a much more accurate representation of the solver variations.

Figure 5 demonstrates how this approach can be used to retrieve the low resolution shape of the performance graph by testing only a small fraction of solvers (in this case 1.5%). More interestingly, while the decision ratios are inaccurate, in this case this information is enough to correctly identify that the most important decision for the N queens problem is the alldifferent propagator and to point out which component is optimal for that decision.

As figure 6 demonstrates, sampling can also be used to improve the effectiveness of iterative deepening based search as well. To achieve this, instead of using the number of total solvers to decide the step sizes, the number of different solvers up to the desired depth has to be used instead.

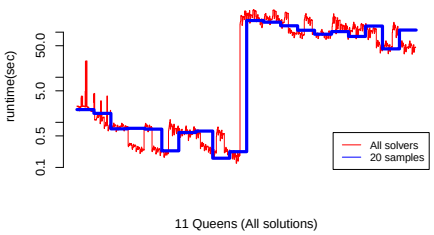


Figure 5: 11 Queens,
20 Samples and All Solvers

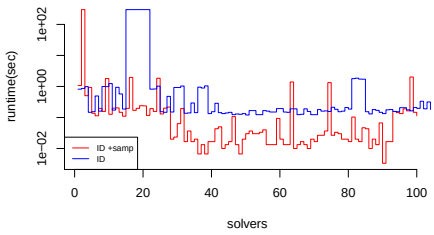


Figure 6: Golomb Ruler,
ID With and Without Sampling

To take this approach one step further, if the total possible solver count is known, by dividing this total by the golden ratio ($\varphi = \frac{1+\sqrt{5}}{2}$) [10] we get a constant number, which can be used to skip the solvers when testing (looping from the start after the last solver is reached). The significance of the golden ratio is that it allows to pick solvers in such way that no solver is picked twice (naturally you have to account for the rounding errors) and at the same time the next solver that is picked this way is always almost at the furthest possible distance from all the previous solvers. Therefore this approach maximizes the variety without knowing how many tests will be done in advance and in turn facilitates using time limits instead of imposing a limit on the number of iterations.

6 Conclusion and Future Work

This paper has presented the current approach for evaluating the performance of individual component configurations. While this approach provides a reasonable starting point for selecting the components for future solvers, it is not yet scalable enough to be applied to more complex constraint problem models. To address this issue, two areas of future study are being investigated: identifying the relationships between the solver components and learning how component performance correspond to the properties of the constraint problem models.

The former approach offers the significant reduction in search space. If it can be determined how component choices affect the performance of other linked components, the isolated components can be tested separately, drastically reducing the number of solvers that need to be tested.

Machine learning, on the other hand, can be used to compare the constraint problem models with previously tested cases and use that information to eliminate a large number of sub-optimal components even before any testing is performed or to modify the search strategy once similar performance patterns are encountered.

Acknowledgements

This work is supervised by Ian Gent and Ian Miguel. Arūnas Prokopas is supported by the Engineering and Physical Sciences Research Council studentship as a part of "A Constraint Solver Synthesizer" project grant.

References

1. Jussien, N., Rochart, G., Lorca, X.: The choco constraint programming solver. In: CPAIOR08 workshop on OpenSource Software for Integer and Contraint Programming OSSICP08. (2008)
2. Wallace, M., Novello, S., Schimpf, J.: Eclipse: A platform for constraint logic programming. ICL Systems Journal 12(1) (1997)
3. Schulte, C., Szokoli, G., Tack, G., Lagerkvist, M., NikoPaltzer, Pekczynski, P.: Gecode reference documentation (2008)
4. Gent, I., Jefferson, C., Miguel, I.: Minion: A fast, scalable, constraint solver. In: IECAI. (2006)
5. Katsirelos, G.: Nogood Processing in CSPs. PhD thesis, University of Toronto (2009)
6. Katsirelos, G., Bachus, F.: Unrestricted nogood recording in csp search. CP (2003)
7. Katsirelos, G., Bacchus, F.: Generalized nogoods in csps. AAAI (2005)
8. Balasubramaniam, D., Jefferson, C., Kotthoff, L., Miguel, I., Nightingale, P.: An automated approach to generating efficient constraint solvers. In: ICSE. (2012)
9. de Silva, L., Balasubramaniam, D.: A model for specifying rationale using an architecture description language. In: 5th European Conference on Software Architecture. (2011)
10. Livio, M.: The golden ratio: the story of phi, the world's most astonishing number. New York: Broadway Books (2002)

A Generic Search Heuristic Based on Survey Propagation to Solve CSPs

Mohamed Ibrahim, Christopher Pal, and Gilles Pesant

École Polytechnique de Montréal, Montreal, Canada
{mohamed.ibrahim, christopher.pal, gilles.pesant}@polymtl.ca

Abstract. Probabilistic message-passing algorithms have had some recent success as the basis for search heuristics to solve satisfaction problems. In particular, Expectation Maximization Survey Propagation (EMSP) has been used in the context of SAT and Expectation Maximization Belief Propagation (EMBP) with both SAT and CSP. In this paper, we propose a new technique inspired from EMSP that both improves search guidance compared to EMBP and generalizes the applicability of EMSP. The contribution is two-fold: first, we derive a generic update rule for EMSP that exploits the structure of the solution space of CSPs; second, we design two original variable and value selection heuristics. Finally we present an empirical comparison of this approach to state-of-the-art search heuristics on several benchmark problems.

Key words. Constraint Satisfaction; Heuristic Search; Expectation Maximization; Survey Propagation

1 Introduction

Constraint Satisfaction Problems often exhibit high complexity, requiring a combination of inference and combinatorial search methods to be solved in a reasonable time. Hsu et al. confirm in [3] that being able to estimate accurately the proportion of solutions featuring a given variable assignment can lead to efficient heuristic search for SAT. Finding such proportion of solutions is the same as computing the marginal (bias) distribution of variables, which is generally intractable. Therefore research efforts have concentrated on approximate methods to estimate such bias distributions.

In this context, probabilistic message-passing algorithms were proven to be very effective as inference methods. For instance, Kask et al. [4] demonstrated the efficiency of BP when applied to CSPs and used it as a value-ordering heuristic. Mezard et al. [2] proposed Survey Propagation (SP) to solve large random Boolean satisfiability problems and emphasized the success of SP in computing marginals proposed later on by Maneva et al. [6]. Hsu et al. [3] suggested Expectation Maximization (EM) variants of these two major message-passing algorithms and, more recently, Le Bras et al. [5] improved EMBP and broadened its applicability. However, EMBP has a main drawback that affects its performance when solving hard CSPs: its underlying BP formulation ignores the probability that a variable might not be the sole support of its constraints, which limits EMBP's ability to exploit the structure of the solution space. In contrast EMSP can capture such information. In this paper, building on [5] and [3], we propose

a novel EMSP-based heuristic that exploit the structure of the solution space using the joker bias in SP to improve heuristic guidance compared to EMBP and adapt it to any CSP.

Basic definitions & Notations

Constraint Satisfaction Problem (CSP). A Constraint Satisfaction Problem (X, D, C) consists of a finite set of *variables* $X = \{x_1, x_2, \ldots, x_n\}$ with respective *finite domains* $D = \{D_1, D_2, \ldots, D_n\}$, together with a finite set of *constraints* $C = \{c_1, c_2, \ldots, c_m\}$. A tuple $s \in D_1 \times D_2 \times \cdots \times D_n$ representing a complete assignment is a solution to CSP iff it satisfies all $c_j \in C$. We will use $\pi_{x_i}(s)$ to denote the value of variable x_i in a solution s and $\#S$ to denote the cardinality of set S.

Close Solutions. Let $\delta(s_1, s_2) = |\{1 \leq i \leq n : \pi_{x_i}(s_1) \neq \pi_{x_i}(s_2)\}|$ denote the *distance* between solutions s_1 and s_2. Identical solutions therefore have distance 0 and we will call two solutions *close* if their distance is at most $\lceil \frac{m}{n} \rceil$, this ratio usually being indicative of the hardness of an instance.

Solution Cluster "κ"***.*** A *solution cluster* is a subset of the solutions such that its elements are pairwise close.

Joker State, Joker Solution. Given a solution cluster κ, we will call "$x_i = *$" a *joker state* of variable $x_i \in X$ to indicate that x_i is unconstrained in κ. Thus x_i can be thought of being in three possible states: either it is constrained to be equal to only one of its values $v \in D_i$ (this means that $x_i = v$ in all solutions of cluster κ), or it is constrained to more than one but not all of its values, or it is unconstrained (this means that in any solution belonging to κ we can replace the value for x_i by any other in D_i and still yield a solution belonging to that cluster). $s^*_{\kappa,i}$ is a *joker solution* in solution cluster κ if it has a variable x_i in a joker state. Such a solution actually stands for all solutions in κ

Bias Distribution. Let S be the set of solutions to (X, D, C). The exact bias distribution of variable x assigned value v, denoted $\Phi_x(v)$, represents the exact proportion of solutions to (X, D, C) where $x = v$:

$$\Phi_x(v) = \frac{\#(\{s \in S : \pi_x(s) = v\})}{\#S}$$

An approximation to that bias distribution will be denoted $\theta_x(v)$.

Joker-Bias Distribution. Let S^* be the set of joker solutions to (X, D, C). The exact joker bias distribution of variable x, denoted $\Phi_x(*)$, represents the exact proportion of joker solutions to (X, D, C) where $x = *$:

$$\Phi_x(*) = \frac{\#(\{s \in S^* : \pi_x(s) = *\})}{\#S^*}$$

An approximation to that bias distribution will be denoted $\theta_x(*)$.

Intuition The main idea is that we do not work any more with individual solutions $s \in S$, but with solution clusters. Some variables may be constrained, i.e. *frozen*, to take a single value inside a solution cluster. It corresponds to the concept of *backbone* variables, but only within the given solution cluster. Other variables may be unconstrained and to handle these one can use the joker state to describe them in the solution cluster and associate it to exactly one generalized value "*". However when we speak about general CSPs, the issue is very difficult as the phase transition, i.e. the structure of the solution space as clusters, of most CSPs is still unknown. Working with joker solutions instead of individual solutions leads us to exploit the structure of the solution space of CSPs even if we don't know it.

2 Expectation Maximization Survey Propagation

EMSP takes advantage of the structure of the solution space and iteratively adjusts the probability for a variable x to be assigned a particular value v for a randomly chosen solution, in addition to a joker bias probability $\theta_x(*)$ for a randomly chosen (joker) solution.

EMSP Update Rule: Let Θ be the vector of variable biases $\theta_{x_i}(v)$, with an additional index for the joker bias $\theta_{x_i}(*)$. Let S correspond to the set of satisfying assignments s *(i.e. unobserved solutions)* and assume that y is a binary-vector variable which indicates the satisfaction of each constraint *(observed)*. Then the goal is to find the variable biases Θ that maximize $logP(y, s|\Theta)$. The E-Step hypothesizes the distribution $Q(s) = P(s|y, \Theta)$. This $Q(s)$ represents the probability of a randomly chosen solution given the biases $\theta_{x_i}(v)$ and the observation y that the constraints are satisfied. The M-step plugs in the estimated $Q(s)$ and uses it to update the bias distribution of the variables. These two steps can be formulated in one general update rule for estimating bias distribution $\theta_{x_i}(v)$, and Joker bias distribution $\theta_{x_i}(*)$ as:

$$\theta_{x_i}(v) = \frac{1}{\tau} \sum_{c \in C : x_i \in X(c)} \left(\sum_{s \in S : x_i = v} Q(s) \right), \; \theta_{x_i}(*) = \frac{1}{\acute{\tau}} \sum_{c \in C : x_i \in X(c)} \left(\sum_{s^* \in S^* : x_i = *} Q(s^*) \right)$$

where S^* is the set of joker solutions and τ, $\acute{\tau}$ are normalizing constants which equal the summation of the numerator over all values of v and summation over all jokers respectively. Note that within this framework so far, the definition of $Q(s)$ and $Q(s^*)$ have remained unspecified. In the following we derive a method that specifies both of them globally.

EMSP **enforcing X-Consistency (**EMSP-G**):** Here, the dependencies between constraints will be considered when computing $Q(s)$, and $Q(s^*)$. In other words the approximation of $Q(s)$ and $Q(s^*)$ will consider every variable in the whole problem. This method enforces "X-Consistency" (domain, bound, or other) on all constraints by considering support tuples that are X-consistent with a variable

assigned a particular value (*constrained case*) and a variable assigned a joker value (*unconstrained case*). Thereby the EMSP-G update rules will be:

$$\theta_{x_i}(v) = \frac{1}{\tau} \prod_{x_j \in X \setminus \{x_i\}} \sum_{v' \in \tilde{D}_j[x_i = v]} \theta_{x_j}(v'), \qquad \theta_{x_i}(*) = \frac{1}{\tilde{\tau}} \prod_{x_j \in X \setminus \{x_i\}} \sum_{v' \in \tilde{D}_j} \theta_{x_j}(v')$$

where $\tilde{D}_j[x_i = v]$, and $\tilde{D}_j$ are the reduced domains of variable x_j after assigning $x_i = v$ and $x_i = *$ respectively while enforcing X-consistency on the whole problem.

3 Deriving a Variable and Value Selection Heuristic

Approximate variable biases are computed at each node of the search tree according to EMSP-G. From that information a two-step search heuristic is derived as variable-value selection. Two variants of the search heuristic are considered:

EMSP ***with maximum joker variable selection ("***EMSP-Max***").*** Select the variable which has maximum joker bias and assign this variable the value which has the maximum bias probability:

$$x_{i'} = \underset{i}{\operatorname{argmax}} \ \theta_{x_i}(*), \qquad v_{j'} = \underset{j}{\operatorname{argmax}} \ \theta_{x_{i'}}(v_j).$$

We then branch on that variable-value pair $x_{i'} = v_{j'}$. The basic idea of EMSP-Max is to guide the search toward the region which has maximum joker solution density or in a solution cluster which has high solution density. Hence for a hard CSP when the solution space consists of several solution clusters, EMSP-Max detects the rich solution cluster in the solution space and guides the search to branch with the variable-value pairs which possess a high solution density in this cluster.

EMSP ***with minimum joker variable selection ("***EMSP-Min***").*** Select the variable which has minimum joker bias (the most likely to fail) and assign this variable the value which has the maximum bias probability (the most likely to succeed):

$$x_{i''} = \underset{i}{\operatorname{argmin}} \ \theta_{x_i}(*), \qquad v_{j''} = \underset{j}{\operatorname{argmax}} \ \theta_{x_{i''}}(v_j).$$

We then branch on that variable-value pair $x_{i''} = v_{j''}$. The joker bias density of the variable measures how the variable is unconstrained in the solution space of the problem. This explicitly shows that the minimum joker bias density in an elegant way adheres to the Fail-First Principle. In other words the less the joker bias of the variable is the more likely search is to fail. Thereby the basic idea of EMSP-Min is to push the search toward the difficult parts of the search tree by selecting the variable which is more likely to fail and then assigning it the value which is more likely to succeed.

4 Experimental Analysis

In order to assess the performance of the two proposed heuristics (EMSP-max, EMSP-min), we performed an experimental analysis on different problems (see [7] & [1] for a description). We compare the proposed heuristics to five generic search heuristics: DomWDeg, IBS, MaxSD, RndVarVal, and closely-related EMBP[5]. Figure 1 plots the number of solved instances vs backtracks and time for the four benchmark problems. A first general observation is that our heuristics perform at least as well and often much better than the related EMBP. Another general remark is that our heuristics do not behave very differently, which may suggest that their value-selection component is the most important here. On the Nonogram and Multi-Knapsack problems, which feature binary variables, EMSP-max and EMSP-min are comparable to state-of-the-art generic search heuristics. On random CSPs with non-binary variables our two heuristics achieve the highest percentage of solved instances (MaxSD is not represented here since there are no structured constraints to use). The usual dominance of our heuristics compared to EMBP confirms the random satisfiability community belief in the success of SP over BP. However on the Rostering problem our heuristics behave poorly, being similar to our baseline random heuristic RndVarVal. This problem features alldifferent constraints, which effectively eliminates the possibility of joker states, a distinctive ingredient in our heuristics.

5 Conclusion

In this paper we proposed generic and efficient CSP search heuristics based on EMSP. Following the approach of [5] our proposal generalized the EMSP framework of [3], enabling it to tackle constraint satisfaction problems. The introduction of joker bias and joker solutions led to exploit the structure of the solution space to the benefit of improving heuristic performance. Preliminary experiments confirm that search heuristics based on EMSP perform better than those based on EMBP and that they are competitive with the state-of-the-art.

References

1. The instances used for these benchmark problems are available http://www.crt.umontreal.ca/∼quosseca/fichiers/20-JAIRbenchs.tar.gz.
2. Braunstein, A., Mézard, M., and Zecchina, R. Survey propagation: an algorithm for satisfiability. *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-2007) cs.CC/0212002* (2002).
3. Hsu, E. I., Muise, C. J., Beck, J. C., and McIlraith, S. A. Applying probabilistic inference to heuristic search by estimating variable bias. In *Proceedings of the 1st International Symposium on Search Techniques in Artificial Intelligence and Robotics (at AAAI08)*. 2008, pp. 68–75.
4. Kask, K., Dechter, R., and Gogate, V. Counting-based look-ahead schemes for constraint satisfaction. In *Proceedings of 10th International Conference on Constraint Programming (CP)* (2004), pp. 317–331.
5. Le Bras, R., Zanarini, A., and Pesant, G. Efficient generic search heuristics within the sf embp framework. In *Proceedings of the 15th international conference*

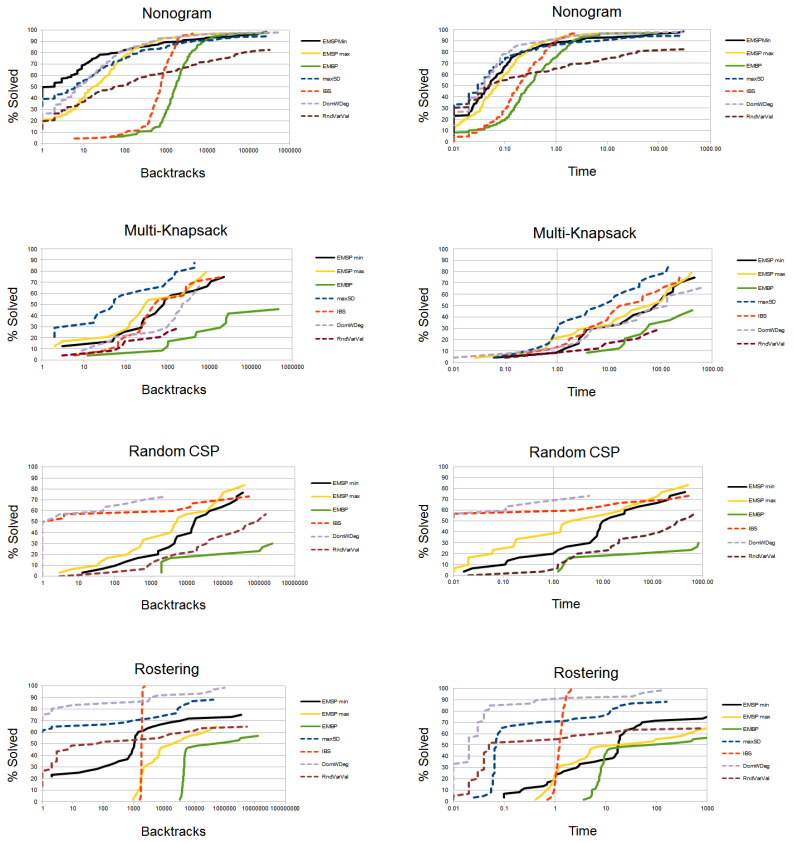


Fig. 1. Percentage of solved instances with respect to the number of backtracks and time for the four benchmark problems. The search heuristics compared are EMSP-max, EMSP-min, maxSD, IBS, EMBP, RndVarVal and domWDeg

.

on principles and practice of constraint programming (Berlin, Heidelberg, 2009), CP'09, Springer-Verlag, pp. 539–553.

6. Maneva, E. N., Mossel, E., and Wainwright, M. J. A new look at survey propagation and its generalizations. *J. ACM 54*, 4 (2007).
7. Pesant, G., Quimper, C.-G., and Zanarini, A. Counting-based search: Branching heuristics for constraint satisfaction problems. *J. Artif. Intell. Res. (JAIR) 43* (2012), 173–210.

A search strategy based on substitutability

Student : Mohamed Rezgui
Director : Jean-Charles Régin
Other Contributor : Arnaud Malapert

I3S Research laboratory
University of Nice-Sophia Antipolis

Abstract. We introduce a new search strategy for enumerating all solutions of a constraint satisfaction problem. This strategy enumerates generic solutions from which all solutions can be efficiently computed. Generic solutions contain substitutable values, that can be replaced by other values in order to generate the set of all solutions. Our strategy favours the appearance of these substitutable values. Consequently, it may save time by generating less generic solutions than solutions. We give some experimental results showing the advantage of this new search for binary CSPs with extensional constraints.

1 Introduction

Constraint Satisfaction problem (CSP) is a formal framework to represent and solve problems in artificial intelligence and operations research. In this paper, we investigate a method for enumerating of all solutions of binary CSP with extensional constraints.
According to the binary CSP definition given by Tsangs in [5], a binary CSP of n variables $[x_1, , x_n]$ has a domain D_i of possible values associated with each variable x_i. Each D_i is finite, and it may not necessarily be the case that all the domains are equal. A binary extensional constraint, C_{ij}, between variables x_i and x_j is a subset of the cartesian product of their domains. Each C_{ij} is finite. We also require that $(a, b) \in C_{ij}$ if and only if $(b, a) \in C_{ij}$. The arity of a constraint C_{ij} corresponds to the number of involved variables. So, binary CSPs contain only constraints with arity as equal as 2.
We propose a new search strategy based upon the concept of substitutability called $Substitutability\ Based\ Search$ (SBS). We consider in this paper only binary CSP. Let Π a binary CSP and Sol_{Pi} a solution of Π. Let (x, a) denote a pair (variable, value) and $ListNoSupports(x, a) = \{(y_1, b_1), ..., (y_k, b_k)\}$, all pairs incompatible with (x, a) (conflicts). Let a and b denote two values of the domain of a variable x, noted $D(x)$. According to [3], a is **substitutable** for b if and only if any solution containing b remains a solution of Π if we replace b by a.
The idea is to detect and favour the appearance of substitutable values to obtain generic solutions, from which we generate all solutions. By enumerating less generic solutions than solutions, we expect to improve efficiency.
Π is a problem involving a variable x, and a and b, two values of $D(x)$, the domain of x. (x, a) is compatible with (y, b) if and only if they can belong to a solution of Π. The value a is substitutable for b if and only if any solution with $x = b$ remains a solution if we replace b by a. More formally, a is substitutable for any value of $D(x)$ if and only if

for any variable y such that a constraint $C(x, y)$ exists and for any variable $b \in D(y)$, (x, a) is compatible with (y, b). Let suppose that a solution with $x = a$, then it is simple to test if a solution with $x = b$ is right. Indeed, it is enough to test the validity of constraints. Let assume that a is substitutable for all other values in $D(x)$. So, with a solution with $x = a$, we can compute the solutions involving other values in $D(x)$. In addition, we can induce the appearance of such substitutable value for any other values of the domain by using a method similar to the algorithm of Bron & Kerbosh [1].
Thus, if we want to induce the substitutability, we need to have compatible values. Let assume that (z, c) is not compatible with (x, a). If we instantiate $z = c$ before $x = a$, then (x, a) becomes substitutable. When (x, a) is substitutable for all remaining values in $D(x)$ then this particular set of values is denoted by $Substitutable\ Set\ of\ Values$ (SSV) otherwise SSV is empty. Our approach allows to induce the apperance of substitutable values during the search. When each substitutable value is found, we save the associated SSV in the $Tuple\ Sequence\ of\ Substitutable\ Elements$ ($TSSE$). TSSE are generic solution but they have some tuples, that they are not solutions. We transform these TSSE to $Global\ Cut\ Seed$ (GCS), introduced by [2], that have only solutions. Finally, we enumerate solutions given by the cartesian product from GCS.
The paper is organized as follows. First of all, let us recall some notions like the substitutability and the common approach of the enumeration of all solutions. Then, we expose the SBS algorithm and an algorithm which computes these GCS. Finally, we present the experimental results giving the pros and cons of SBS.

2 Classical enumeration of solutions

The search algorithm for enumerating all solutions on a binary tree is defined as follows. First, a variable x and a value a are chosen. $x = a$ is instantiated and then, we propagate the consequences of this instantiation. In the presence of failure (any variable of the CSP Π with an empty domain), the search backtracks and branches with $x \neq a$. When taking a decision ($x = a$ or $x \neq a$), it creates a choice point and a node of the search tree is created. The search uses strategies to select the pair (variable, value) at each choice point. After each choice point, the process is repeated. When all variables are instantiated, the search has found a solution and then a backtrack occurs.

3 SBS : Search Based Substitutability

3.1 SBS algorithm

Definition 1 *A* **Tuple Sequence of Substitutable Elements** *(TSSE) is an n-tuple of substitutable elements* $(\{a_1, SSV(a_1)\}, ..., \{a_n, SSV(a_n)\})$*, where each substitable element is a nonempty set of values containing a value a, which is substitutable for all values containing in the associated* **Substitutable Set of Values** *(SSV).*

SBS algorithm (1) induces the apperance substitutable values to obtain generic solutions. It computes generic solutions from which all solutions will be calculated. Whereas the classical algorithm selects a variable and then successively tries each value

Algorithm 1: Search Based Substitutability

```
1  SBS(X)
2  (x, a) ← getVariableValueMinConflicts()
3  for each (y, b) ∈ ListNoSupports(x, a) do
4      TSSE[y] ← TSSE[y] ∪ {b}
5      saveStateAndAssign(y, b)
6      if propagate(y = b) then
7          SBS(X)
8      restoreStateAndUnassign(y, b)
9      TSSE[y] ← TSSE[y] − {b}
10     if not propagate(y ≠ b) then
11         return
12 TSSE[x] ← TSSE[x] ∪ D(x)
13 saveStateAndAssign(x, a)
14 if propagate(x = a) then
15     SBS(X)
16 restoreStateAndUnassign(x, a)
17 TSSE[x] ← TSSE[x] − D(x)
```

of this variable for each level of the search tree, SBS proceeds differently. First of all, in line 2 (1), SBS identifies the pair (x, a), which has the least conflicts. Note that we use the minconflicts heuristic described by Minton & Steven in [4] for variable and value selection to get the pair (x, a). Afterwards, SBS first instantiates conflicts of (x, a), from Line 3 to 11 (1). When no conflicting value remains, a is a substitutable value for x, and we can instantiate x to a. At that time, other values of $D(x)$ do not need to be instantiated with x because (x, a) is substitutable for them. When we instantiate (x, a), the algorithm saves the remaining values of $D(x)$ in $SSV(a)$, then it saves in a $TSSE$, in Line 4 and 12 (1) . Finally, we instantiate (x, a). Once all variables are instantiated, $TSSE$ becomes a generic solution. Then, we are going to compute GCS from $TSSE$. We detailed in the next section.

3.2 Enumeration of Global Cut Seed

Definition 2 *[2] A* **Global Cut Seed** *(GCS), is an n-tuple $\Delta = (\delta_1, ..., \delta_n)$, where each δ is a nonempty set of values, such that each n-tuple $(v_1, ..., v_n), v_n \in \delta_1, ..., \delta_n$ and $v_i \in D_n$ is a solution of the problem Π. The value n corresponds to the length of the GCS.*[1]

We can enumerate the solutions in GCS, which are given by the cartesian product of the domain. Thus, we propose to transform each TSSE into a set of GCS. Let assume that $TSSE_i$ that contains a value Sub that is compatible with all other values $TSSE_j$

[1] We considered a part of the definition described by [2], because we want to reduce the notion of GCS only a list of tuples, that have compatible values.

with $j > i$.
A $TSSE$ can have incompatible values. So, we generate GCS, that have only compatible values.
We detail this transformation in the following example. For instance, let a CSP with five variables and $TSSE = [\{\underline{a_1}, b_1\}, \{\underline{a_2}, b_2, c_2\}, \{\underline{c_3}\}, \{\underline{d_4}\}, \{\underline{e_5}, f_5, g_5\}]$. Underlined values correspond to the substitutable values. Constraints of Π define the following incompatibilities: e_5 incompatible with b_2 and c_2, g_5 incompatible with b_1 and c_3 incompatible with b_1. Let $GCS[i]$ denote all values of the variable x of the GCS. We use a backward algorithm, that maintains the following property: at each end of the step i, it creates a tuple S_i contains all compatible values sets from x to x_n. S_i corresponds to a $Global\ Cut\ Seed$ of length $n - i$. Indeed, at $i = 0$, the algorithm creates a $Global\ Cut\ Seed$ with compatible values of the last instantiated variable. Afterwards, at each step i, it creates S_i of $Global\ Cut\ Seed$ from $Global\ Cut\ Seed$ created in step $i - 1$. For that, it adds a compatible value if and only if there are nonempty sets after elimination of non-compatible values. To compute all solutions contained in $TSSE$, we generate from the last substitable element the first GCS $[\{e_5, f_5, g_5\}]$, then at each step $i - 1$, it checks the compatibility of each value of all $TSSE_i$ with values examing with previous sets. Thus, at step 4, it adds the set $\{d_4\}$, which gives S_4 containing $[\{d_4\}, \{e_5, f_5, g_5\}]$. It continues the same way for the list S_3, so we have $[\{c_3\}, \{d_4\}, \{e_5, f_5, g_5\}]$. In step 2, e_5 is incompatible with b_2 and c_2. So S_2 has 3 GCS: $[\{a_2\}, \{c_3\}, \{d_4\}, \{e_5, f_5, g_5\}]$, $[\{b_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$, $[\{c_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$. Finally, in step 1, g_5 is incompatible with b_1 and c_3 is incompatible with b_1. So S_1 contains:
$[\{a_1\}, \{a_2\}, \{c_3\}, \{d_4\}, \{e_5, f_5, g_5\}]$,
$[\{a_1\}, \{b_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$,
$[\{a_1\}, \{c_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$,
$[\{b_1\}, \{b_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$,
$[\{b_1\}, \{c_2\}, \{c_3\}, \{d_4\}, \{f_5\}]$.
The GCS $[\{b_1\}, \{a_2\}, \{\}, \{d_4\}, \{e_5, f_5, g_5\}]$ is not valid because it contains an empty set. After generating five GCS in step 1, we use the cartesian product in order to enumerate all solutions from this generic solution. Finally, there are ten solutions:
$[\{a_1\}, \{a_2\}, \{c_3\}, \{d_4\}, \{e_5\}]$, $[\{a_1\}, \{a_2\}, \{c_3\}, \{d_4\}, \{f_5\}]$,
$[\{a_1\}, \{a_2\}, \{c_3\}, \{d_4\}, \{g_5\}]$, $[\{a_1\}, \{b_2\}, \{c_3\}, \{d_4\}, \{f_5\}]$,
$[\{a_1\}, \{b_2\}, \{c_3\}, \{d_4\}, \{g_5\}]$, $[\{a_1\}, \{c_2\}, \{c_3\}, \{d_4\}, \{f_5\}]$,
$[\{a_1\}, \{c_2\}, \{c_3\}, \{d_4\}, \{g_5\}]$, $[\{b_1\}, \{b_2\}, \{c_3\}, \{d_4\}, \{f_5\}]$,
$[\{b_1\}, \{b_2\}, \{c_3\}, \{d_4\}, \{g_5\}]$, $[\{b_1\}, \{c_2\}, \{c_3\}, \{d_4\}, \{f_5\}]$.
So, the compression factor is 2, because from 5GCS, we obtain 10 solutions. Note that a new GCS can have identical suffix than a previous one. For instance, we have two GCS for S_2, $[\{b_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$ and $[\{c_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$ which share $[\{c_3\}, \{d_4\}, \{f_5, g_5\}]$. In this case, the algorithm calls the function MERGEEQUIVALENTGCS, that merges identical GCS. For that, it transforms the GCS in a sequence of values. Then, it sorts this sequence by lexicographic order to determine if there are equalities between GCS and then it concatenates identical GCS. The complexity of our algorithm is as follows. The cost of lexicographic order is O($n \log(n)$) comparisons. Each comparison costs the sum over the values of the GCS. In our case, we have d con-

sidered elements and the size of the maximal string is nd. So, the cost is O($nd^2 \log(d)$). In our example, we have after concatenation of the two GCS $[\{b_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$ and $[\{c_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$, an only one GCS $[\{b_2, c_2\}, \{c_3\}, \{d_4\}, \{f_5, g_5\}]$. We can try to concatenate GCS in several different ways, but the enumeration of unique solutions (which is mandatory) may become difficult.

4 Results

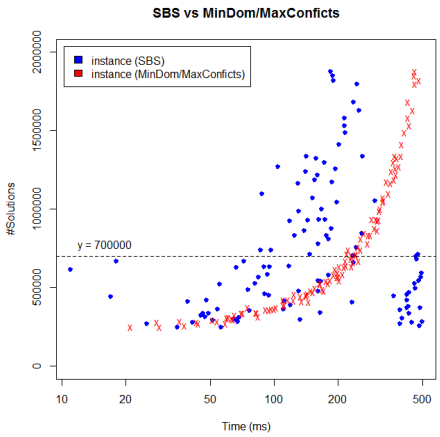


Fig. 1. Comparison of solving times

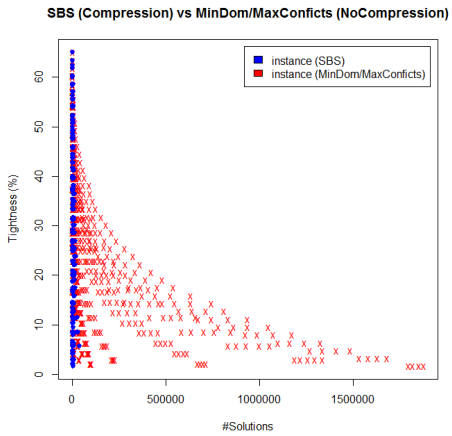


Fig. 2. Comparison of compression solutions

We compare the SBS approach to the classical enumeration scheme. Then, we evaluate the impact of the GCS concatenation. All experiments were performed on an ad-hoc solver with an Intel Core i7 quad core (2.2 GHz). Each experiment is run on a single core. Instances are generated randomly by setting a number of variables V, a number of values per domain D, a number of extensional constraints C, and a tightness T, which defines the number of unauthorized tuples for each constraint C. Note that problems with a low tightness have many solutions, whereas problems with a high tightness have less solutions. Let $MinDom$ denote a variable selection heuristic, that selects the variable x with the smallest domain size. Let $MinConflict$ and $MaxConflict$ denote two value selection heuristics, that select the value of a of $D(x)$ having respectively the minimum and maximum conflict. We test SBS with the different selection heuristics $MinDom/MinConflict$ and $MinDom/MaxConflict$. Experiments show no significant differences in solving time between $MinDom/MinConflict$ and $MinDom/MaxConflict$ over all solved instances. Thus, we consider only the comparison between SBS and $MinDom/MaxConflict$. In the following figures, each instance solved is represented by a point. We checked that the number of solutions found for each instance is exactly the same between SBS and $MinDom/MaxConflict$.

Figure 1 compares the solving times of each instance. The scatter plot associated the strategy $MinDom/MaxConflict$ forms a curve. Solving times of instances solved by SBS are scattered between the bottom and top of the curve for $MinDom/MaxConflict$. It appears that below 700,000 solutions, SBS is sometimes faster and sometimes slower than $MinDom/MaxConflict$. However, SBS clearly outperforms the other method above 700,000 solutions.
Figure 2 compares the number of generic solutions for SBS and the number of solutions as function as the tightness. The number of generic solutions generated by SBS varies from 10 to 37 000. The compression factor of solutions with SBS is significant when the tightness is low (less than 25 %). Indeed, the compression factor varies between 2 and 50 (number of solutions under the number of compressed solutions). As far as the tightness increases, the compression factor decreases. There is no compression from 50 % of tightness. Thus, determining the tightness of the problem is very important in order to choose in advance an appropriate search strategy to enumerate all solutions.

5 Conclusion

We proposed a new search strategy based on the substitutability (SBS) to compute all solutions of a binary CSP with extensional constraints. According to the experimental results, SBS enhances the problems with a factor 2-5 for solving time and a factor 2-50 in terms of memory. However, this approach becomes less effective when the $tightness$ increases. In further work, we will test on real world problems and try to generalize this idea to non-binary CSP.

References

1. C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communications of the ACM*, pages 575–577, 1973.
2. Focacci and Milano. Global cut framework for removing symmetries. *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 77–92, 2001.
3. Eugene C. Freuder. Eliminating interchangeable values in csp. *Proceedings of the ninth National conference on Artificial intelligence - Volume 1*, pages 227–233, 1991.
4. Steven. Minton and Ames Research Center. The min-conflicts heuristic [microform] : experiment and theoretical results. *The min-conflicts heuristic [microform] : experiment and theoretical results*, page 1 v., 1991.
5. Edward Tsang. Foundations of constraint satisfaction, 1993.

A constraint programming based approach for planning milk runs

Anne Meyer

FZI Forschungszentrum Informatik, 76131 Karlsruhe, Germany,
meyer@fzi.de,
WWW home page: http://www.fzi.de

Abstract. Milk runs – a logistics concept of lean manufacturing – can be modelled as a Period Vehicle Routing Problem (PVRP) with additional requirements on regularity and robustness. We propose a CP formulation for PVRPs based on an existing VRP model, which can be easily extended. Furthermore, we suggest a CP based Parallel Insertion Heuristic to construct feasible starting solutions and a Large Neighborhood Search as a local search improvement technique. For the latter we present different adaptions to the PVRP case, amongst them a new relatedness measure improved with regard to (client specific) side constraints.

Keywords: constraint programming, period vehicle routing problem, lean supply chain management

1 Introduction

Lean manufacturing has had, and still has today, a massive impact on the way of producing discrete goods all over the world since the early nineties, when Steven Womack's book "The machine that changed the world" appeared. The key idea is to reduce variability in all processes, since variability means "waste". To preserve this reduced variability up stream in a supply chain, regularly scheduled and levelled transports should be established. These regular transports are called milk runs, and are a widely accepted and natural logistical concept within the lean manufacturing theory (cmp e.g. [1]). Outside of the "lean world", there are even more vehicle routing problems (VRP), where similar aspects as transparency, regularity, ease of control, etc. play a more and more important role (see amongst others [10]). Furthermore, a practical planning system has to be tailored to every customer, since – as in every VRP – customers have different requirements on the objective function and the constraints considered.

2 A CP model for Periodic Vehicle Routing Problems

A milk run planning problem belongs to the class of Period Vehicle Routing Problems (PVRP). For the basic formulation a complete network graph with

corresponding arc costs, a planning period in days, a depot node, a set of customer nodes and a set of vehicles with fixed capacity is given. The objective is, to serve the demand of all customers considering their allowed schedules (for example mon - wed, tue - thu, wed - fri) and the capacity restrictions of the vehicles, while minimizing the cost of the travelled distance (see [2]). So, the PVRP involves three simultaneous decisions: Selecting a feasible schedule for every client node, assigning visits to vehicles for every day, and route the vehicles on every day.

The introduced constraint model for PVRP is an extension of the Vehicle Routing Problem with Time Windows of [5]. It mainly relies on the concept of path constraints. A path is built by the variable P_i representing the direct predecessor of a visit i. In order to get a stronger propagation, all path constraints are formulated redundantly once for P_i and once for S_i, representing the direct successor of the visit i.

For PVRPs we have to adapt the vector of visits: If a client order has a frequency request of three it is represented by three visits. Each vehicle is represented by a first and a last visit for every day of the planning horizon, on which it is available. The resulting vector $\mathcal{V}$ for an instance with three customer orders $\mathcal{C}$, with frequency requests of three, two and one, and a planning period of three days with two vehicles available looks as follows:

$$\mathcal{V} = [\overbrace{\underbrace{1,2,3}_{\mathcal{C}_0},\underbrace{4,5}_{\mathcal{C}_1},\underbrace{6}_{\mathcal{C}_2}}^{\text{customer visits } \mathcal{C}},\overbrace{\underbrace{7,8}_{\mathcal{M}_0},\underbrace{9,10}_{\mathcal{M}_1},\underbrace{11,12}_{\mathcal{M}_2}}^{\text{first (vehicle) visits } \mathcal{F}},\overbrace{13,14,15,16,17,18}^{\text{last (vehicle) visits } \mathcal{L}}]$$

The newly introduced variable H_i represents the day when a visit i is serviced.

$$H_{P_i} = H_i \qquad i \in \mathcal{C} \cup \mathcal{L} \qquad (1)$$

$$H_{S_i} = H_i \qquad i \in \mathcal{C} \cup \mathcal{F} \qquad (2)$$

$$H_i < H_{i+1} \qquad h \in \mathcal{O};\quad i \in \mathcal{C}_h;\quad i+1 \in \mathcal{C}_h \qquad (3)$$

$$H_i \leqq t \quad \Leftrightarrow \quad V_i \leqq \max(\mathcal{M}_t) \qquad i \in \mathcal{C}; t \in \mathcal{T} \qquad (4)$$

$$H_i \geqq t \quad \Leftrightarrow \quad V_i \geqq \min(\mathcal{M}_t) \qquad i \in \mathcal{C}; t \in \mathcal{T} \qquad (5)$$

This information is maintained along the route by path constraints (1) and (2), line (3) breaks symmetries. The last two groups of constraints (4) and (5) link vehicles and days ($\mathcal{M}_t$ represents the available vehicles on day t).

In a milk run use case, the days of service should be evenly distributed over the planning horizon. For a frequency request of two and a horizon of four the allowed schedules would be $\mathcal{A}_h = [[0,2],[1,3]]$.

$$\bigwedge_{0 \leq j < f_h} (H_{\mathcal{C}_h^j} = s_a^j) = pat_a \qquad s_a^j \in \mathcal{A}_h;\quad a \in \{0 \dots |\mathcal{A}_h|\};\quad h \in \mathcal{O} \qquad (6)$$

$$\exists! a \in \{0 \dots \mathcal{A}_h\} : pat_a = true \qquad h \in \mathcal{O} \qquad (7)$$

For the example above, the boolean variable pat_0 in constraint (6) indicates if schedule 0 is assigned for order h. Line (7) ensures that there is exactly one pattern, which is assigned.

In accordance with lean principles it is preferable, that clients are serviced at roughly the same time, i.e. that the difference between the start of service T of two visits of the same order is smaller than a parameter L. Within the VRP literature this is called time consistency (see [3]) and it can be expressed by the following constraints:

$$T_i - T_j \leqq L \wedge T_j - T_i \leqq L \qquad i, j \in \mathcal{C}_h \wedge i < j \tag{8}$$

Apart from that, other requirements might come up in milk run use cases: It could be requested that a customer is always serviced by the same driver or that the tour duration is limited due to legal regulations on working hours. Another important aspect within lean production is a levelled use of resources, that is, a levelled use of vehicles per day. Due to the modelling power and flexibility of CP, most of the additional constraints or changes in the objective function can be modelled in a rather natural way. However, for the solution approaches it is a special challenge to be robust against these additional constraints or even better to exploit them actively.

3 Heuristics and first computational results

Almost all successful approaches for larger VRP instances are heuristics made up of a construction and an improvement phase. As former studies could show, an insertion based construction heuristic (IH) and a Large Neighborhood Search (LNS) for the improvement phase, both based on Constraint-based techniques, are promising approaches for VRPs with additional side constraints (see e.g. [4] or [6]).

Parallel Insertion Heuristic for PVRP. In this type of construction heuristics, visits are inserted step by step into an emerging plan. It can be characterized first by the order, in which the visits are inserted and second if the insertions are sequential or parallel. In the parallel case there is more than one route at a time, into which visits can be inserted (see e.g. [8]). We implemented two simple sorting strategies for the PVRP: (1) *FrequDom*: We sort the orders by frequency (decreasing) and the domain size of P (increasing) of the first visit of the order. That is, we start the insertion process with the visits of an order with daily service request, whose domain of P of the first visit is most restricted. (2) *FrequTheta*: We sort the orders by frequency (decreasing) and by the angle θ of the polar coordinates of every customer (like in a Sweep Algorithm). For the insertion we either search for the best position of all visits of an order by a CP based tree search or by an *InsertBrancher*, which tries to insert the visits of the current order at the insertion position where the least detour is caused.

Since, the visits of the same order have to be served on different days, we automatically "open" different tours on different days and have a parallel version of the heuristic. Furthermore, we post an additional constraint, making sure that – if there is still a free vehicle for the insertion candidate – the distance to the direct predecessor of the insertion position has to be smaller or equal than the distance to the depot, if not, a new route is "opened" and the routes tend to be more compact.

Large Neighborhood Search. LNS is a local search improvement technique based on the idea, to take out repeatedly visits of a feasible tour plan and reinsert them at (hopefully) better positions. The way how visits are selected and the way how they are re-inserted defines the neighborhood. In order to avoid a situation, where the re-insertion decisions of the selected visits are completely independent and degenerate to many little reinsertions, it is useful to remove visits which are somehow related. A very simple strategy is to select visits, which are geographically close (radial select). But in a periodic VRP it is not necessarily helpful, since the visits might be geographically close to each other but have to be visited on different days. Also, Shaw's relatedness measure, often re-used in literature, does not seem to be suitable, since it merely favours intra route improvements (for details see [9]).

Therefore, we introduced a new relatedness measure with a simple idea: The more feasible neighbors two visits have in common, the more related they are, that is the more interrelated are the reinsertion decisions. For calculating this measure, we build a set of the k nearest, feasible predecessors of every visit i (from the domain of P_i in the root node) and we count the number of joint neighbors for every pair of visits. This number is our relatedness measure. The advantage of this domain based measure is, that additional constraints, like special vehicle requirements, service on certain days, etc., are captured automatically.

We use the relatedness measure within two different selection schemes: For the *RandomRelatedSelect* scheme, we randomly choose a seed node and select the n most related visits with regard to the seed node. The *RandomRelatedSameDaySelect* considers only the most related visits, which are assigned to the same day in the current solution, in order to get an intra day improvement. Apart from the latter, we implemented a second PVRP specific selection scheme, the *RandomOrderSelect*, where we select all visits of one or more random orders. Thereby, we are able to select another allowed schedule for the order during the reinsertion step. The reinsertion step for all seven implemented selection schemes (the others are common schemes from literature) is performed by a CP based tree search with a first improvement stopping criterion.

First Computational Results. In our first computational experiments[1] we tested the heuristics on PVRP benchmark instances from [2]. These instances incorpo-

[1] The application is implemented with Gecode 3.7.3 (built with VS 2010), a free C++ software library for developing constraint-based systems. All experiments were executed on the following machine: Intel Core 2 DUO E8400, 3.00 GHz, 4.0 GB RAM, 64 bit windows operating system.

rate time windows and a maximum tour duration and the size ranges from 100 up to 1000 visits. In order to be able to measure the solution quality, we did not consider any additional milk run specific constraints at this stage of the work.

With all versions of the parallel IH we achieved the goal to produce feasible starting solutions for at least 17 or rather 18 out of the 20 instances with a quality between 15% and 100% above the best known solutions (see also table 1). In case of the infeasible solutions there are left over 1-8 visits (or 1-3 orders), but the resulting tour plan is feasible with regard to all constraints and we expect that this small number of unplanned stops can be easily inserted during the insertion based improvement phase. Without loosing a lot of solution quality compared to the search for the best insertion positions, we showed that it is possible to use the *InsertBrancher*, which saves up to a factor of 7 on the runtime. For the smallest instance the solution time is around a second, whereas, for the largest it takes up to 600 seconds. A comparison with initial solutions of the literature on specialized algorithms for the PVRP-TW is difficult, since there is no run time or quality indication. It is only reported that starting solutions violate time windows, capacity and tour length restrictions. Considering the expected problem sizes in a milk run environment and the level of sophistication of the implementation, the performance of the IH is satisfying for us.

Table 1. Overview on solution quality of the parallel insertion heuristic by sorting strategy in the first column and insertion strategy in the first row (avg. dev. BKS: average deviation from best known solution, infeasible: number of infeasible instances with the total number of missing visits of these instances in brackets)

	Best insertion		*InsertBrancher*	
	avg. dev. BKS	infeasible	avg. dev. BKS	infesaible
FrequDom	56%	2 (5)	59%	2 (10)
FrequTheta	50%	3 (9)	51%	3 (10)

For the LNS phase we tested the performance of the neighborhoods by conducting randomly chosen neighborhoods, accepting only improving solutions and stopping this process after a certain time limit is reached. We could show, that all neighborhoods contribute to the improvement of the solution, but that there is none dominating the others across different instances and across different phases of the search. The only neighborhood performing very well on all instance during all phases is the described *RandomRelatedSameDaySelect*. This shows the need of a meta-heuristic steering the selection of neighborhoods depending on the instance and the progress of the improvement phase.

4 Conclusions and Outlook

To sum up, it can be stated that the choice of CP based heuristics proved successful, since the solution quality and the run times for PVRP-TW are satisfying and

the flexibility of the model with respect to milk run and client specific extensions looks promising. The experiments for the LNS show that our new relatedness measure works well, but that there is the need of a meta-heuristic to direct the selection of neighborhoods. Therefore, the next step will be the implementation of an Adapted Large Neighborhood Search (see [7]), since this is a natural and very successful extension of the LNS concept, especially, in the area of VRP. Furthermore, we will implement more milk run specific extensions and test the robustness of the algorithms with respect to these extensions. Thereby, the most challenging tasks will be to deal with uncertain and correlated demand and with the requirement that – in case of a necessary re-planning process of the milk plan – the established regularities should not be destroyed.

References

1. Baudin, M. (2004). *Lean Logistics: the nuts and bolts of delivering materials and goods*.
2. Cordeau, J.-F., G. Laporte, A. Mercier (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society 52*, 928–936.
3. Groër, C., B. Golden, and E. Wasil (2008). The consistent vehicle routing problem. *Manufacturing and Service Operations Management Articles in Advance*, 1–14.
4. Kilby, P., P. Prosser, and P. Shaw (2000). A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Constraints 5*(4), 389–414.
5. Kilby, P. and P. Shaw (2006). Vehicle routing. In *Handbook of Constraint Programming*, Chapter 23, pp. 799–834. Francesca Rossi and Peter van Beek and Toby Walsh.
6. Nickel, S., M. Schröder, and J. Steeg (2012). Mid-term and short-term planning support for home health care services. *European Journal of Operational Research 219*, 574–587.
7. Pisinger, D. and S. Ropke (2010). Large neighborhood search. In M. Gendreau and J.-Y. Potvin (Eds.), *Handbook of Metaheurisitcs* (2. ed.)., pp. 399–419. Springer.
8. Potvin, J.-Y. and J.-M. Rousseau (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research 66*(3), 331 – 340.
9. Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. *Principles and Practice of Constraint Programming–CP98*, 417–431.
10. Smilowitz, K., M. Nowak, and T. Jiang (2012). Workforce management in periodic delivery operations. *Transportation Science, Forthcoming*, 1–17. Published Online ahead of print March 8, 2012.
11. Womack, J. P., D. T. Jones, and D. Roos (1990). *The Machine That Changed the World: The Story of Lean Production*.