GAC on Conjunctions of Constraints

George Katsirelos[1] and Fahiem Bacchus[1]

Department of Computer Science, University Of Toronto,[*]
Toronto, Ontario, Canada
[gkatsi|fbacchus]@cs.toronto.edu

Abstract. Applying GAC on conjunctions of constraints can lead to more powerful pruning [1]. We show that there exists a simple heuristic for deciding which constraints might be useful to conjoin. The result is a useful automatic way of improving a CSP model for GAC solving.

1 Introduction

Generalized arc consistency (GAC) [2] is arc consistency generalized to the non-binary case. GAC can be used as a mechanism for constraint propagation to reduce the complexity of a CSP. GAC can also be embedded in a backtracking search algorithm to provide constraint propagation dynamically during search as done in the binary case with in the MAC algorithm [3, 4].

Bessière and Régin have pointed out GAC can also be applied to conjunctions of constraints [1], and they provide an algorithm for enforcing GAC on conjunctions. In this paper we make some further observations on this idea. Specifically, we show that there is a simple heuristic that can be used to determine whether or not it might be worthwhile grouping a collection of constraints into a conjunction and performing GAC on this conjunction.

This provides a method for achieving a potential improvement of any given CSP model automatically: using the heuristic we simply group together collections of constraints, performing GAC on these conjunctions instead of on the individual constraints. The model with conjoined constraints can then be tested to determine if in fact an improvement has been achieved. Importantly, performing GAC on a conjunction of constraints does not require modifying the original model: the original representation of the constraints in each conjunction can be used directly by the GAC algorithm. It simply requires identifying the sets of constraints that might be worth conjoining. Since CSP modeling is a complex and potentially costly task, any improvements to a model that can be achieved automatically can help reduce the cost of finding a sufficiently efficient model. We demonstrate that this idea can be effective using a two different CSP models.

2 Background and Notation

A CSP $(\mathcal{V}, \mathcal{C})$ consists of a set of variables $\mathcal{V} = \{V_1, \ldots, V_n\}$ and a set of constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$. Each variable V has a finite domain of values *Dom*$[V]$, and can be assigned a value v, indicated by $V \leftarrow v$, if and only if $v \in$ *Dom*$[V]$. Let $\mathcal{A}$ be any set of assignments. No variable can be assigned more than one value, so $||\mathcal{A}|| \leq n$ (i.e., the cardinality of this set is at most n). When $||\mathcal{A}|| = n$ we call $\mathcal{A}$ a *complete* set of

[*] This research was supported by the Canadian Government through their NSERC program.

assignments. Associated with $\mathcal{A}$ is a set of variables $\mathit{VarsOf}(\mathcal{A})$: the set of variables assigned values in $\mathcal{A}$.

Each constraint C is over some set of variables $\mathit{VarsOf}(C)$, and has an arity equal to $\|\mathit{VarsOf}(C)\|$. A constraint is a set of sets of assignments: if the arity of C is k, then each element of C is a set of k assignments, one for each of the variables in $\mathit{VarsOf}(C)$. We say that a set of assignments $\mathcal{A}$ *satisfies* a constraint C if $\mathit{VarsOf}(C) \subseteq \mathit{VarsOf}(\mathcal{A})$ and there exists an element of C that is a subset of $\mathcal{A}$.

$\mathcal{A}$ is said to be *consistent* if it satisfies all constraints C such that $\mathit{VarsOf}(C) \subseteq \mathit{VarsOf}(\mathcal{A})$, i.e., it satisfies all constraints it fully instantiates. Otherwise it is inconsistent. A *solution* to a CSP is a complete and consistent set of assignments.

Definition 1 (Generalized Arc Consistency). *Given a constraint C and a variable $V \in \mathit{VarsOf}(C)$, a value $a \in$ Dom$[V]$ is **supported** in C if there is a set of assignments $\mathcal{A} \in C$, such that $V \leftarrow a \in \mathcal{A}$. $\mathcal{A}$ is called a **support** for $\{V \leftarrow a\}$ in C. C is **(generalized) arc consistent** iff each value a of each variable $V \in \mathit{VarsOf}(C)$ is supported in C. The entire CSP is **arc consistent** iff each of its constraints is arc consistent.*

Definition 2 (Conjunctive Consistency (Bessière and Régin)). *Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be a set of constraints and $\mathit{VarsOf}(\mathcal{C}) = \cup_i \mathit{VarsOf}(C_i)$ be the set of variables these constraints are over. $\mathcal{C}$ is conjunctively GAC iff for each value a of each variable $V \in \mathit{VarsOf}(\mathcal{C})$ there exists a set of assignments $\mathcal{A}$ such that (1) $V \leftarrow a \in \mathcal{A}$, (2) $\mathit{VarsOf}(\mathcal{A}) \supseteq \mathit{VarsOf}(\mathcal{C})$, and (3) $\mathcal{A}$ is consistent. We call such an $\mathcal{A}$ a **support** for $\{V \leftarrow a\}$ on the conjunction $\mathcal{C}$.*

Intuitively, this definition says that there exists a *single* assignment extending $V \leftarrow a$ that is consistent with all of the constraints in $\mathcal{C}$.

As shown in [1], conjunctive GAC can be achieved in time upper bounded by $O(d^{\|\cup_i \mathit{VarsOf}(C_i)\|})$ where d is the maximum domain size of any of the variables in $\mathit{VarsOf}(\mathcal{C})$. Furthermore, their method does not require computing an explicit representation of the conjunction. That is, the representation of the individual constraints contained in the original model can be used directly.

Consider a CSP model containing the two constraints C_1 and C_2. If GAC is going to be used to solve the CSP, we would have to perform GAC(C_1) and GAC(C_2), a task that would require time $O(d^{\|\mathit{VarsOf}(C_1)\|})$ plus $O(d^{\|\mathit{VarsOf}(C_2)\|})$ [5]. If we choose to perform GAC on their conjunction instead, GAC($C_1 \wedge C_2$), we would need time $O(d^{\|\mathit{VarsOf}(C_1) \cup \mathit{VarsOf}(C_2)\|})$. That is, the increased time required decreases as the number of variables shared by C_1 and C_2 increases. In the extreme case where $\mathit{VarsOf}(C_1) \subseteq \mathit{VarsOf}(C_2)$ performing GAC on the conjunction instead of over each constraint individually has the same order of complexity. In fact, for the GAC-schema algorithm of [1] it would be faster to compute GAC($C_1 \wedge C_2$) that computing both GAC(C_1) and GAC(C_2) in this case—each of these computations would require approximately the same amount of time, so we reduce the number of separate GAC computations by doing GAC on the conjunction.

From the definition it can be seen that GAC($C_1 \wedge C_2$) is a stronger consistency condition than individually enforcing GAC(C_1) and GAC(C_2). For the separate application of GAC to the two constraints, we would require only that there exists two sets of assignments $\mathcal{A}_1$ and $\mathcal{A}_2$ both extending $V \leftarrow a$, with $\mathcal{A}_1$ satisfying C_1 and $\mathcal{A}_2$ satisfying

C_2. GAC on the conjunction requires that there exist a single set of assignments $\mathcal{A}_\wedge$ extending $V \leftarrow a$ and satisfying C_1 and C_2 simultaneously. Hence, GAC($C_1 \wedge C_2$) has the potential to prune more values.

What is also interesting, but was not as well highlighted in [1], is that the relative strength of GAC on the conjunction increases as the number of shared variables between C_1 and C_2 increases. If these two constraints share no variables, then we can find the required set of assignments $\mathcal{A}_\wedge$ satisfying the conjunction, by simply unioning the two sets of assignments $\mathcal{A}_1$ and $\mathcal{A}_2$. That is, in this case GAC on the conjunction is identical to GAC on each constraint separately. On the other hand if these two constraints share i variables, then the required set of assignments $\mathcal{A}_\wedge$ only exists if we can find an $\mathcal{A}_1$ satisfying C_1 and $\mathcal{A}_2$ satisfying C_2 that agree on their assignments to all i shared variables (and then $\mathcal{A}_\wedge$ can again be $\mathcal{A}_1 \cup \mathcal{A}_2$). Clearly, this constraint on the individually satisfying assignments becomes stronger as the number of shared variables, i, grows.

Hence, the optimal case is when $\mathit{VarsOf}(C_1) \subseteq \mathit{VarsOf}(C_2)$: GAC($C_1 \wedge C_2$) is faster and it yields the maximal improved strength over doing GAC on each constraint separately.

In general, a simple heuristic for grouping constraints into conjunctive sets is as follows.

1. Initialize CS, the set of conjunctive sets, to contain m conjunctive sets each containing the single constraint C_i.
2. If there exist two conjunctive sets $\mathcal{C}_1, \mathcal{C}_2 \in CS$ such that (1) $||\mathit{VarsOf}(\mathcal{C}_1 \cup \mathcal{C}_2)|| \leq max(||\mathit{VarsOf}(\mathcal{C}_1)||, ||\mathit{VarsOf}(\mathcal{C}_2)||) + M$ remove $\mathcal{C}_1$ and $\mathcal{C}_2$ from CS and add $\mathcal{C}_1 \cup \mathcal{C}_2$.
3. Repeat 2 until no more such pairs exist.

In this algorithm M places a limit on the increase in arity we are willing to allow. An optimal value for M will be problem dependent. For example, if $M = 0$ the algorithm will only conjoin constraints satisfying $\mathit{VarsOf}(C_1) \subseteq \mathit{VarsOf}(C_2)$. We could also place restrictions on the minimal number of shared variables.

3 Empirical Results

We report on experiments we performed with two different CSP models. The first is a model for the Golomb ruler problem containing quaternary constraints developed in [6]. In this model, there is a quaternary constraint $|x_i - x_j| \neq |x_k - x_l|$ for all marks $x_1, \ldots, x_m, j < i, l < k$. Hence, over every set of four marks there will be 7 different constraints posted over these variables, 4 quaternary constraints and 3 ternary constraints. We consider finding optimal solutions and proving them to be optimal using a backtracking search that maintains GAC. For this model we set $M = 0$, thus we only conjoin constraints whose variables are a subset of another.

The results are shown in Table 1. We used a 500MHz PIII machine, and the same underlying implementation for GAC on all of our tests. The first column shows the number of branches explored (and time required in CPU seconds) using the constraints without any conjunctions. The second column shows what happens when we conjoin together only the 3 quaternary constraints, and the final column show what happens when we conjoin all 7 constraints (for each set of 4 variables). The results show that a useful improvement is speed is obtained and a moderate improvement in the number of branches

Fig. 1 Backtracks and cpu time to find (F) a golomb ruler of a given size or prove (P) its optimality. “-” indicates that the solver was unable to find a solution after reaching 10^5 backtracks.

Problem size	goal	Quat $\cup$ Tern		($\wedge$ Quat) $\cup$ Tern		$\wedge$ (Quat $\cup$ Tern)	
7	F	95	0.14	95	0.07	90	0.06
7	P	988	1.62	988	0.81	894	0.51
8	F	574	1.33	574	0.64	492	0.45
8	P	7791	27.51	7791	13.35	7131	7.99
9	F	5581	26.51	5581	12.53	4920	7.75
9	P	57545	403.28	57545	190.16	52868	117.16
10	F	40141	360.64	40141	167.45	36666	107.24
10	P	-	-	-	-	-	-

explored. The middle column demonstrates that our technique is heuristic—improved pruning is not always obtained through conjunctions. In particular, it turns out that the three quaternary constraints all logically entail each other, so enforcing one is as powerful as enforcing all three or as enforcing the conjunction. Thus, we see no improvement in number of branches. The improvement in speed arises from the fact that with conjunctions we have fewer constraints over which GAC has to be enforced.

In the second model we experiment with random 3-sat problems. These problems are converted from a set of clauses into a CSP containing a set of binary variables and a ternary constraint for each clause. In this case, we set $M = 1$, thus generating conjunctions over 4 variables. The results are shown in Table 2.

Fig. 2 Average number of backtracks and cpu time to prove whether a problem is satisfiable or not. The last column indicates the percentage of instances where the solver performed better if constraints were conjoined.

		avg leafs		avg time		
# Variables	# Instances	original	w/conjunctions	original	w/conjunctions	perc
60	100	674.33	522.71	0.9951	0.8485	71
70	100	1637.36	1300.21	2.8882	2.4676	74
80	100	3504.36	2888.33	7.204	6.3182	74
90	100	9204.49	7085.21	21.7589	17.8116	75
100	100	19573.2	14434.2	52.5704	40.7247	83

It is worthwhile noting in this case that there were instances where the solver actually performed more backtracks when using conjoined constraints than it did in the original problem. This anomaly can be attributed to the fact that 3-SAT has a special structure, which is not accounted for by the minimal remaining values-break ties by degree heuristic used. Therefore, even though the conjunctive constraints cause more pruning, they end up making the search slower by fooling the heuristic. This anomaly could probably be eliminated by using one of the heuristics that have been developed specifically for SAT problems.

If we factor out these instances, and look only at those instances where the solver performed fewer backtracks with conjunctive constraints, we find that the cpu time used is at worst only 10% more than the time used to solve the problem using the original model. This shows that despite the fact that we are generating constraints of higher arity in this case (and hence the complexity of enforcing GAC on the conjunction is higher than enforcing GAC on the individual constraints separately) the overhead of performing GAC on the conjoined constraints is alleviated by the extra pruning it generates. More specifically, Table 3 show the same results as Table 2 except that the instances where the heuristic performed poorly have been removed. The results show a useful improvement is achieved by using conjunctions.

Fig. 3 The same data Table 2 but only instances for which conjoining constraints did not interfere with the behavior of the DVO heuristic are counted.

		avg leafs		avg time		
# Variables	# Instances	original	w/conjunctions	original	w/conjunctions	perc
60	79	1602.62	540.81	2.35886	0.879747	88.6076
70	78	3890.79	1322.47	6.84756	2.50756	93.5897
80	82	7829.66	2706.33	16.0824	5.94561	90.2439
90	79	21926.1	7214.34	51.7696	18.1687	94.9367
100	85	44843.2	15398.1	120.359	43.482	97.6471

References

1. C. Bessière and J.-C. Régin. Local consistency on conjunctions of constraints. In *Proceedings of the ECAI'98 Workshop on Non-binary constraints*, pages 53–59, Brighton, UK, 1998.
2. A. K. Mackworth. On reading sketch maps. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 598–606, Cambridge, Mass., 1977.
3. J. Gaschnig. Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for satisficing assignment problems. In *Proceedings of the Second Canadian Conference on Artificial Intelligence*, pages 268–277, Toronto, Ont., 1978.
4. D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 125–129, Amsterdam, 1994.
5. C. Bessière and J.-C. Régin. Arc consistency for general constraint networks: Preliminary results. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 398–404, Nagoya, Japan, 1997.
6. B. Smith, K. Stergiou, and T. Walsh. Using auxilliary variables and implied constraints to model non-binary problems. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, Austin, Texas, 2000.