Postponing Branching Decisions

Willem Jan van Hoeve[1] and **Michela Milano**[2]

Abstract. Solution techniques for Constraint Satisfaction and Optimization Problems often make use of backtrack search methods, exploiting variable and value ordering heuristics. In this paper, we propose and analyse a very simple method to apply in case the value ordering heuristic produces ties: **postponing the branching decision**. To this end, we group together values in a tie, branch on this sub-domain, and defer the decision among them to lower levels of the search tree. We show theoretically and experimentally that this simple modification can dramatically improve the efficiency of the search strategy.

1 BACKGROUND

Constraint Satisfaction Problems (CSPs) and Constraint Optimization Problems (COPs) are defined on a set of variables $x_1, \ldots, x_n$ representing problem entities. Variables range on finite domains $D_1, \ldots, D_n$ and are subject to a set of constraints C that define the feasible configurations of variable-value assignments. A COP in addition has an objective function to be optimized. A solution to a CSP or a COP is a variable-value assignment respecting all constraints, and optimizing the objective function if present. When being solved with Constraint Programming, the solution process interleaves constraint propagation and search.

A general way of building a search tree for solving CSPs and COPs, called *labelling*, consists in selecting a variable and assigning it a single value from its domain. The variable and value selection are guided by heuristics, such that the most promising variable/value is selected first. If the value selection heuristic regards two or more values equally promising we say the heuristic produces a tie. The definition of ties can be extended to the concept of *heuristic equivalence* [1] that considers equivalent all values that receive a rank within a given percentage from a value taken as reference. A similar situation occurs when different domain value heuristics are applied simultaneously. In general, ties occur when the used heuristic(s) define(s) a *partial order* on the values' ranks. In such situations, labelling chooses a single value according to a deterministic (f.i. lexicographic order) or a randomized (see [1]) rule.

We propose a very simple, yet extremely effective method that improves the efficiency of tree search in presence of ties: avoid making the choice and postpone the branching decision. This is done by grouping together equivalent values in a sub-domain and performing a branching on the whole sub-domain, while those that are clearly ranked by the heuristic are still assigned singularly to a variable. We call this method *partitioning*. The resulting search tree may consist of a sub-problem at depth n, which must be searched further. In this paper, we will always search the sub-problem via labelling. This means that the leaves of the search tree appear at depth between n and $2n$.

Although labelling and partitioning have been used in practice before, to our knowledge no theoretical or experimental comparison has been made thus far.

[1] CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands email: W.J.van.Hoeve@cwi.nl

[2] DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy email: mmilano@deis.unibo.it

2 THEORETICAL COMPARISON

Similar to the analysis of limited discrepancy search (LDS) by Harvey and Ginsberg [3], we introduce a probability (distribution) of the heuristic being successful. Given that the current search state has a solution in its subtree (i.e. it is a *good* state), the heuristic probability is the probability that the d-th choice of the heuristic is also a good state. For a state corresponding to variable x_i and choices $d \in D_i$, we denote this probability by p_i^d. A leaf l of a search tree consists of the instantiation of all n variables, i.e. $l = \{x_i = d_{i_j} \mid d_{i_j} \in D_i, i \in \{1, \ldots, n\}\}$. The probability (with respect to the heuristic) of a leaf l being successful is

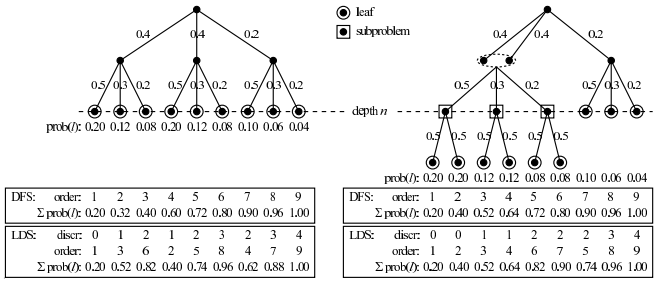
$$\mathrm{prob}(l) = \prod_{\{x_i = d_{i_j}\} \in l} p_i^{d_{i_j}}.$$

In Figure 1 we compare labelling and partitioning on trees corresponding to 2 variables, both having 3 domain values. The branches are ordered from left to right following the heuristic's choice. The heuristic probability of success is shown for each branch. Note that the heuristic produces a tie, consisting of two values, for the first variable. Labelling follows the heuristic on single values, while partitioning groups together values in the tie. For depth-first search (DFS) and LDS, the order in which the leaves are visited is given, together with the cumulative probability of success. When we apply LDS to partitioning, we only count the discrepancies in the upper part (depth 0 to n) of the tree. Note that for every leaf, partitioning always has a higher (or equal) cumulative probability of success than labelling. This is formalized in Theorem 1.

Theorem 1 *For a fixed variable ordering and a domain value ordering heuristic, let T_{label} be the search tree defined by labelling, and let $T_{\mathrm{partition}}$ be the search tree defined by partitioning (grouping together ties), both without constraint propagation. Assume $p_i^{d_1} > p_i^{d_2}$ if the heuristic prefers d_1 over d_2 for $d_1, d_2 \in D_i$. Let the set of the first k leaf nodes visited by labelling and partitioning be denoted by L^k_{label} and $L^k_{\mathrm{partition}}$ respectively. If T_{label} and $T_{\mathrm{partition}}$ are both traversed using DFS or LDS*[3]*, then*

$$\sum_{l \in L^k_{\mathrm{partition}}} \mathrm{prob}(l) \geq \sum_{l \in L^k_{\mathrm{label}}} \mathrm{prob}(l). \qquad (1)$$

[3] In fact, Theorem 1 holds for any 'depth-first based' search strategy.



a. labelling b. partitioning

Figure 1. Cumulative probability of success using DFS and LDS.

Proof. Omitted due to space restrictions, but available in an extended version of the paper [8]. □

3 EXPERIMENTAL COMPARISON

We have compared partitioning and labelling on the Travelling Salesman Problem (TSP) and the Partial Latin Square Completion Problem (PLSCP), implemented on a Pentium 1Ghz with 256 MB RAM, using ILOG Solver 5.1 [5] and Cplex 7.1 [4].

For the TSP, we have used a constraint programming model and a heuristic similar to [6] based on reduced costs. For this problem, the heuristic is very accurate, and propagation is rather poor. Subproblems are being solved using DFS, since all leaves can be considered to have equal probability of being successful. We stop the search as soon as an optimal solution has been found. The results of our comparison are presented in Table 1. The instances are taken from TSPLIB [7]. For labelling and partitioning, the table shows the time and the number of fails (backtracks) needed to find an optimum. For labelling, the discrepancy of the leaf node that represents the optimum is given. For partitioning, the discrepancy of the sub-problem that contains the optimum is reported. On all instances but one, partitioning performs much better than labelling. Observe that for the instance 'dantzig42' labelling needs less fails than partitioning, but uses more time. This is because partitioning solves the sub-problems using DFS, and lacks the LDS overhead.

	labelling			**partitioning**		
instance	time (s)	fails	discr	time (s)	fails	discr
gr17	0.08	36	2	0.02	3	0
gr21	0.16	52	3	0.01	1	0
gr24	0.49	330	5	0.01	4	0
fri26	0.16	82	2	0.01	0	0
bayg29	8.06	4412	8	0.07	82	1
bays29	2.31	1274	5	0.07	43	1
dantzig42	0.98	485	1	0.79	1317	1
swiss42	6.51	2028	4	0.08	15	0
hk48	190.96	35971	11	0.23	175	1
brazil58	N.A.	N.A.	N.A.	0.72	770	1

N.A. means 'not applicable' due to time limit (900 s).

Table 1. Results for finding optima of TSP instances (not proving optimality).

For the PLSCP we have used a straightforward constraint programming model, using *alldifferent* constraints on the rows and the columns, with maximal and effective propagation. As heuristic we have used a simple, not very accurate, first-fail principle for the values, i.e. values that are most constrained are to be considered first. The sub-problems of partitioning are again being solved using DFS. For both labelling and partitioning, constraint propagation is applied throughout the whole search tree. We made a comparison on balanced and unbalanced instances at the phase transition of the problem, generated with the PLS-generator [2]. The name of an instance, 'b/u.on.hm', indicates an (un)balanced instance of order n with m holes. The results are reported in Table 2, which follows the same format as Table 1. Clearly, partitioning has a negative impact on the propagation. Although partitioning often outperforms labelling due to the lack of LDS overhead, the decreased propagation is too important to neglect. Hence none of the methods always outperforms the other on these PLSCP instances.

	labelling			**partitioning**		
instance	time (s)	fails	discr	time (s)	fails	discr
b.o25.h238	2.36	668	5	1.09	746	5
b.o25.h239	0.49	15	1	0.42	2	1
b.o25.h240	1.17	179	4	0.86	893	4
b.o25.h241	3.31	772	3	4.70	3123	4
b.o25.h242	2.41	537	3	1.80	1753	4
b.o25.h243	4.06	1082	4	3.96	2542	4
b.o25.h244	1.33	214	3	2.99	2072	4
b.o25.h245	9.40	2308	6	10.66	12906	7
b.o25.h246	2.01	401	5	2.22	1029	4
b.o25.h247	258.91	69105	6	11.66	5727	4
b.o25.h248	33.65	6969	5	0.68	125	2
b.o25.h249	212.76	60543	11	101.46	85533	8
b.o25.h250	2.45	338	2	0.83	687	3
u.o30.h328	273.53	32538	4	82	14102	3
u.o30.h330	21.79	2756	3	25.15	5019	3
u.o30.h332	235.40	30033	5	56.94	9609	3
u.o30.h334	4.18	256	2	6.09	843	2
u.o30.h336	1.73	69	2	0.76	12	1
u.o30.h338	49.17	5069	3	29.41	8026	3
u.o30.h340	1.68	91	2	0.81	66	2
u.o30.h342	28.40	3152	3	5.41	600	2
u.o30.h344	9.05	605	2	8.35	1103	2
u.o30.h346	2.15	101	2	3.76	482	2
u.o30.h348	43.80	2658	2	32.86	2729	2
u.o30.h350	1.16	46	1	0.80	12	1
u.o30.h352	5.10	288	2	0.95	32	1
sum	1211.45	220793	91	396.62	159773	81
mean	46.59	8492.04	3.50	15.25	6145.12	3.12

Table 2. Results for PLS completion problems.

REFERENCES

[1] C.P. Gomes, B. Selman, and H. Kautz, 'Boosting Combinatorial Search Through Randomization', in *Proceedings of the Fifteenth National Conference on Artifi cial Intelligence (AAAI'98)*, pp. 431–437, (1998).

[2] C.P. Gomes and D. Shmoys, 'Completing Quasigroups or Latin Squares: A Structured Graph Coloring Problem', in *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, (2002).

[3] W. D. Harvey and M. L. Ginsberg, 'Limited Discrepancy Search', in *Proceedings of the Fourteenth International Joint Conference on Artifi cial Intelligence (IJCAI-95)*, volume 1, pp. 607–615, (1995).

[4] ILOG. ILOG Cplex 7.1, Reference Manual, 2001.

[5] ILOG. ILOG Solver 5.1, Reference Manual, 2001.

[6] M. Milano and W.J. van Hoeve, 'Reduced cost-based ranking for generating promising subproblems', in *Eighth International Conference on the Principles and Practice of Constraint Programming (CP'02)*, volume 2470 of *LNCS*, pp. 1–16. Springer Verlag, (2002).

[7] G. Reinelt, 'TSPLIB - a Traveling Salesman Problem Library', *ORSA Journal on Computing*, **3**, 376–384, (1991).

[8] W.J. van Hoeve and M. Milano. Postponing branching decisions, 2004. http://homepages.cwi.nl/∼wjvh/papers/postpone.pdf.