Using Uncertainty in Backtracking

Steven Prestwich

Cork Constraint Computation Centre
Department of Computer Science
National University of Ireland at Cork
s.prestwich@cs.ucc.ie

For problems in which completeness is not needed we are faced with a choice of search algorithm type, two common candidates being stochastic local search (SLS) and systematic backtracking (BT). SLS often out-performs BT on large problems, with much better scalability on problems such as solvable random 3-SAT from the phase transition. However, BT performs very well on many problems where its constraint-handling ability is crucial. Both BT and SLS have strengths and weaknesses, and research has recently been devoted to combinations of the two.

Several approaches combine some of SLS's scalability with BT's completeness and/or constraint handling, using clever backtracking strategies or learning techniques to avoid redundant computations. However, there is currently no search algorithm (to my knowledge) that combines complete and non-redundant search, low and bounded memory usage, and scalability demonstrably equal to that of SLS. Such an algorithm can be seen as a holy grail of this area of research, but it may not be possible even in principle. Perhaps the goals of fast single-solution search and fast complete search are incompatible and should be tackled by different forms of algorithm.

A useful approach to hybridising two algorithms is to try unusual combinations of their features. Uncertainty is one feature of SLS that distinguishes it from BT. It makes randomised moves in both directions: forward towards a solution or local minimum, and backward via the use of noise. Is uncertainty an important feature of SLS, and can it be used to improve BT's scalability? It can of course be combined with BT in the forward direction: the selection of variables and values can contain a random element, though too much randomness reduces the benefits of ordering heuristics. My view is that uncertainty can profitably be combined with BT in the *backward* direction. That is, the choice of backtracking variable is made non-systematically: randomly or using a heuristic with random tie-breaking. I will refer to this randomised form of backtracking as RB.

Flexibility in the choice of backtracking variable has previously been shown to be important (Partial Order Dynamic Backtracking) and RB simply takes this to an extreme: total flexibility at the cost of completeness. Perhaps surprisingly, it turns out to have SLS-like scaling on hard combinatorial problems such as SAT, graph colouring, maximum cliques, Golomb rulers and N-queens. Its advantage over SLS is that it retains constraint handling abilities such as domain pruning, allowing it to solve problems that are unsuitable for SLS yet too large for BT. But why does it scale like SLS? My view, which may not convince everyone (feedback is welcome!), is that it *is* SLS but in a different space: the consistent partial assignments. Where constraint propagation is combined with RB, the space is the subspace of partial assignments that are consistent under propagation (no domain wipe-out). In other words, RB combines the search space of BT algorithms with the search strategy of SLS. If this view is correct then the question *why does RB scale well?* is reduced to *why does SLS scale well?* — still something of an open question. This view also blurs the distinction between BT and SLS, because RB appears to be both at the same time.

However, there are some tricky implementation issues. A pragmatic way of implementing RB is to pick an assigned variable randomly and backtrack to it, using the standard chronological backtracking machinery in our favourite constraint language. The other variables that were unassigned can then be reassigned, and their values restored where this causes no conflict. In this way RB can be combined with techniques such as forward checking, arc consistency, and more specialised constraints such as all-different or cumulative constraints. I have not tested this implementation thoroughly, but in tests it did not always give good results. It is hard to see why this is so, but my guess is that, in SLS terms, it is too noisy. If RB is indeed SLS then its objective function (to be minimised) is the number of unassigned variables. This implementation often unassigns a large number of variables, which is equivalent to making large backward moves: in other words, high noise. For some problems high noise is necessary, but for others it is too disruptive. Still, the implementation is worth exploring further because of its ability to exploit existing constraint systems.

I have been taking an alternative implementation approach, combining RB with an interesting feature of Dynamic Backtracking: its ability to unassign a variable without unassigning those assigned since. RB can then unassign a small number (for example 1) of randomly-chosen variables without affecting others, which is cheap and (I claim) corresponds to low-noise SLS. The drawback with this approach is that it is harder to implement and cannot easily be combined with an existing constraint system based on chronological backtracking. Some mechanism must be found for efficiently updating the state of the system when unassigning an arbitrary assigned variable. This is easy to achieve with branch-and-bound-style cost constraints, and it can be done with forward checking on both binary and non-binary constraints. Future work will extend RB to other forms of constraint handling.

Another future aim is to solve problems that currently require more complex hybrid approaches in which BT and SLS interact but are kept distinct. Such hybrids have given impressive results, but intuitively it seems likely that a more unified algorithm should do better. Finally, a complete RB would be nice, but as an aim of RB is to solve very large problems it should ideally not be based on memory-intensive learning techniques.