Bounds-Consistent Local Search*

Student: **Stefania Verachi**
Supervisor: **Steven Prestwich**

Cork Constraint Computation Centre
Department of Computer Science
University College, Cork, Ireland
{s.verachi,s.prestwich}@cs.ucc.ie

Abstract. This paper describes a hybrid approach to solving large-scale constraint satisfaction and optimization problems. It describes a hybrid algorithm for integer linear programming which combines local search and bounds propagation, inspired by the success of a randomized algorithm for Boolean Satisfiability (SAT) called Unit-Walk. A dynamic prioritization heuristic has been developed to improve the algorithm, inspired by another algorithm called Squeaky Wheel Optimization.

1 Introduction

Several algorithms have been proposed to solve both Constraint Satisfaction Problems (CSPs) and optimization problems. Basically two class of algorithms have been used to solve specifically the two classes of problems: complete and constructive algorithms such as backtrackers, and random and incomplete ones such as local search algorithms. Backtrackers are complete and therefore always find a solution or prove the unsolvability. They can also exploit the structure of the problem by using constraint propagation, improving search efficiency. Unfortunately, they sometimes scale poorly to large problems [6].

Local search algorithms are typically non-constructive as they search a space of total but inconsistent assignments, but they often scale up much better to large problems. A drawback of these algorithms is that they usually do not exploit pruning techniques, making them uncompetitive on some highly-structured problems. However, it is possible to make them more efficient on such problems by incorporating constraint propagation techniques [7].

The aim of our project is to study and solve hard large-scale problems by using a new class of algorithms that combines techniques such as local search, constraint propagation and relaxation. Our first hybrid algorithm combines a form of local search with a simple but powerful form of constraint reasoning: bounds consistency applied to linear inequalities on integer variables. We are developing this hybrid by applying it to Multiple Sequence Alignment (MSA) problems from Bioinformatics, for which we have an Integer Linear Programming (ILP) model [8].

* This work has been supported by Science Foundation Ireland under Grant 04/BR/CS0355.

2 Preliminaries

Most incomplete randomized algorithms for SAT are local search algorithms [1] and they often perform well on optimization problems. A local search algorithm chooses an initial configuration at random and then modifies it step by step. Usually the configuration is a total variable assignment, but in some cases [6, 1, 2] (including our new algorithm) partial assignments are used.

The form of constraint reasoning we chose is Bounds Consistency (BC) on ILP models, for several reasons:

- ILP is a simple, well-known and expressive constraint language that has been used for decades to model optimization problems. In particular, it subsumes SAT. We therefore expect it to be sufficient for modelling our problems.
- BC on linear inequalities is equivalent to hyper-arc consistency, or arc-consistency (AC) on non-binary CSPs, which should be sufficiently powerful for a wide range of applications.
- When solving very large problems, memory becomes an important issue. Enforcing BC requires only two values to be maintained for each integer variable: an upper and a lower bound. In contrast, maintaining AC on arbitrary constraints requires maintaining information for every variable domain value, which may become impractical for problems with many variables and large domains. In [10] it has been shown that using a weaker propagation method as BC does not increase the size of the search space when the domains contain no "holes".

An efficient BC algorithm for linear constraints is given in [11]. Note that BC can also be applied to non-linear constraints such as all-different, and we plan to implement some of these constraints in future work.

To combine BC with local search, our first idea was to use *Incomplete Dynamic Backtracking* (IDB) [6]. IDB has previously been combined with forward checking for some (binary and non-binary) CSPs, and with arc-consistency for binary CSPs. It uses a technique similar to those used for Dynamic CSPs to maintain variable domains during randomized search. However, we believe that combining IDB with BC on linear inequalities will require a complex implementation that uses a great deal of memory, so instead we turn to another approach used in at least two previous algorithms: Unit-Walk and Squeaky Wheel Optimization. Both of these behave like backtrackers until a dead-end is reached, then they restart the search but use information from previous iterations. This is a less incremental approach than IDB, but has given good results.

2.1 The UnitWalk algorithm

UnitWalk [1] is a simple algorithm that performs well on different instances of hard SAT problems. It combines local search and unit clause elimination, and gave best results in the "industrial problem class" in a recent SAT competition. It also has been proved to be Probabilistically Approximately Complete (PAC),

meaning that as time tends to infinity the probability of finding a solution tends to 1. This implies that, at least in principle, random restarts are not required.

The Unit-Walk procedure is divided into *periods*. During one period at least one flip is made. A period starts with choosing a random permutation of variables. Then the algorithm takes the input formula and modifies it step by step, sometimes also modifying the current assignment. At each step, the algorithm substitutes the value of one variable in the current formula. If there are unit clauses, then the variable v is taken from one of them; if the value of v does not satisfy the unit clause and satisfies no other unit clause, it is flipped before the substitution. If there are no unit clauses, the algorithm substitutes the value to the next variable in the chosen permutation (taking the value from the current assignment). If a period finishes (i.e, all variables are processed), but no variable was flipped during it, the algorithm chooses a variable at random and flips it (in fact, this is a very rare situation). After a period finishes, the algorithm chooses a new random permutation, replaces the current formula by the input formula, and starts a new period. After a MAXPERIODS number if no satisfying assignment is found, the random walk restarts from another random initial assignment. And if the algorithm fails after MAXTRIES periods, it terminates.

2.2 Squeaky Wheel Optimization

The basic idea of the Squeaky Wheel Optimization algorithm [2] is to restart the search whenever a dead-end is reached. But now an informed search runs every time. A greedy algorithm, with no backtracking, is used to construct a solution. This "solution" may violate hard constraints. Its failure yields information about the "trouble spots". This information is used to adjust the variable ordering in the next iteration, via dynamic variable priorities. The "trouble makers" thus get high priority, a version of the well-known "fail-first" technique.

3 The Algorithm

We have developed a slightly different version of Unit-Walk, and we are still testing other options. At the beginning the algorithm, like Unit-Walk, starts with a random value sequence that we shall call the *suggested assignments* for the variables is generated, and a random permutation of the variables to be used as a variable ordering. First it applies BC to the problem (terminating if inconsistency is detected), possibly reducing the domain sizes. It then proceeds by selecting a variable (guided by the permutation), assigning a value to it (guided by the suggested value), and propagating bounds. If the suggested value is not currently in the domain then another value must be chosen; we are currently experimenting with heuristics to guide these aspects of the algorithm.

Unlike in Unit-Walk, if domain wipe-out occurs during propagation (not only on reaching a value for which all possible assignments which cause wipe-out) the search restarts with a new permutation, with the troublesome variables promoted to a higher position in the permutation (as in Squeaky Wheel Optimization), but

with the same suggested assignments. This contrasts with another randomized algorithm called Iterative Sampling proposed in [4], which starts the new search by using new random assignments. However, sometimes the algorithm, like most local search algorithms, can become trapped in an endless loop. In order to avoid this, a new random value for each variable is generated after N iterations if no one flip occurs during a search. N is tuned depending on the specific problem. A similar technique is used in Unit-Walk.

3.1 Multiple Sequence Alignment

The algorithm aims to solve large-scale optimization problems, one example being Multiple Sequence Alignment (MSA). Aligning DNA or protein sequences is one of the dominant problem in computational molecular biology. The task of comparing several sequences simultaneously has been formalized in different ways.

Initially researchers compared pairs of DNA or protein sequences to check whether or not they were homologues. The number of all possible alignments for two sequences of length N is equal to $\frac{2^{2N}}{\sqrt{2N\pi}}$. Thus the complexity of the problem is exponential in the problem size. The MSA problem is NP-complete, and a challenging combinatorial problem for researchers interested in both Constraint Programming (CP) and Operations Research (OR) [8].

Dynamic programming (DP) was the first approach to the problem in order to find an optimal alignment, but is known to scale poorly to more than a few sequences. Kececioglu [3] introduced in 1991 a formulation of the problem for multiple sequences, based on a graph-theoretic approach that can be modelled as an ILP problem. This is called the *Complete Maximum Weight Trace* (CMWT) formulation. The symbols are viewed as vertices in an *alignment graph* $G = (V, E)$. Each vertex (i, j) denotes a position i in sequence j. An edge e is a connection between two vertices; each edge has a weight w representing the value of the alignment between two symbols. The set of the realized edges is a *trace*. The goal is to maximize the total weight.

The problem with only two sequences is solved by DP in polynomial time, but the number of edges rapidly increases as the number of sequences and their lengths grow. Another formulation exploits this sparsity: the *Sparse Maximum Weight Trace* (MWT). In this formulation only some sub-graphs are used in the model, and the meaningful sub-graphs are chosen in order to reduce the size of the complete model.

With the CMWT or MWT formulations, branch-and-bound and branch-and-cut algorithms have been proposed to solve the ILP to optimality [9, 5]. A linear model of MSA, called a *hybrid ILP model* has been proposed in [8]. The problem is to maximize $\sum w_\epsilon v_\epsilon$, with $\epsilon \in E$, and where v_ϵ is a binary variable and w_ϵ is the *weight* of each edge ϵ, representing the usefulness of aligning its two symbols, as in the MWT formulation. This model has two sets of variable: integer variables pij representing the position of the symbol in the alignment matrix, and boolean variables v_ϵ saying whether an alignment is realized or not.

Two sets of linear constraints are employed: *ordering constraints*: $p_{ij} - p_{i'j'} \geq 1$ to maintain the order of symbols in each sequence, and *channelling constraints*: $p_{ij} - p_{i'j'} + cv_\epsilon \leq c$, $p_{i'j'} - p_{ij} + cv_\epsilon \leq c$ to relate position variables p_{ij} and boolean variables v_ϵ. We are currently testing the algorithm and we hope that our new bounds-consistent local search approach will be well-suited to this linear MSA model.

4 Conclusion

The work described is still in progress. The aim is to find new hybrid methods to solve constrained optimization problems, with an emphasis on large structured problems such as those encountered in real applications. We call our approach *bounds-consistent local search*. It combines a local search algorithm (related to previous algorithms such as Unit-Walk, Squeaky Wheel Optimization and Iterative Sampling) with bounds consistency on integer linear inequalities.

At the time of writing, we are developing our prototype algorithm, using examples of MSA and Social Golfer problems. Results obtained so far are encouraging but the heuristics require further experimentation. Experiments were made on some instances of the Social Golfer problem. The algorithm solves a few cases such as 2 groups of 2 golfers for 3 weeks, which has 42 variables and 78 constraints. On MSA examples it takes longer than was expected. We are trying to improve these results by experimenting with heuristics for both the variable permutation and the set of suggested assignments.

In future work we shall improve both the heuristics and the implementation, in order to speed up the search. We will also extend the algorithm to non-linear constraints such as all-different, enabling other problems to be modelled effectively; bounds consistency can be applied to such constraints. We will also experiment with alternative models for the MSA problem, and investigate other interesting problems.

References

1. E. A. Hirsch, A. Kojevinkov. UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. In *PDMI preprint 9/2001, Steklov Institute of Mathematics at St.Petersburg,* 2001.
2. D. E. Joslin, D. P. Clements. "Squeaky Wheel" Optimization. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98).* Madison, WI, 1999.
3. J. D. Kececioglu. Exact and Approximation Algorithms for DNA Sequence Reconstruction. PhD thesis, University of Arizona, 1991.
4. P. Langley. Systematic and Nonsystematic Search Strategies. *First International Conference on Artificial Intelligence Planning Systems*, 1992.
5. S. Minton, A. Philips. Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems. et al. *Artificial Intelligence.* 1993.
6. S. Prestwich. Local Search and Backtracking vs Non-Systematic Backtracking. *AAAI 2001 Fall Symposium on Using Uncertainty within Computation. To appear.*

7. S. Prestwich. Combining the Scalability of Local Search with the Pruning Techniques of Systematic Search. *Annals of Operations Research (to appear)*.
8. S. Prestwich, D. Higgins, O. O'Sullivan. A SAT-Based Approach to Multiple Sequence Alignment. *Poster, Ninth International Conference on Principles and Practice of Constraint Programming, Kinsale, Ireland*, 2003, pp. 940-944.
9. K. Reinert, H. P. Lenhof, P. Mutzel, K. Mehlhorn, J. D. Kececioglu. A Branch-and-Cut Algorithm for Multiple Sequence Alignment. *Proceedings of the 1st Annual International Conference on Computational Molecular Biology (RECOMB)*. 1997.
10. C. Schulte, P. J. Stuckey. When Do Domain Propagation Lead to the Same Search Space. *Transactions on Programming Languages and Systems. ACM Press, (to appear)*. 2005.
11. Y. Zhang, R. H. C. Yap. Arc Consistency on n-ary Monotonic and Linear Constraints. In *Proceedings of CP-00*, pp. 470-483, Singapore, 2000.