Extending Systematic Local Search for Job Shop Scheduling Problems

Bistra Dilkina, Lei Duan, and William S. Havens

Intelligent Systems Laboratory, Simon Fraser University
Burnaby, British Columbia V5A 1S6, Canada
{bnd,lduan,havens}@cs.sfu.ca

1 Introduction

Hybrid search methods synthesize desirable aspects of both constructive and local search methods. Constructive methods are systematic and complete, but exhibit poor performance on large problems because bad decisions made early in the search persist for exponentially long times. In contrast, stochastic local search methods are immune to the tyranny of early mistakes. Local search methods replace systematicity with stochastic techniques for diversifying the search. However, the lack of systematicity makes remembering the history of past states problematic. Typically, hybrid methods introduce a stochastic element into a basically constructive search framework. Lynce [6] uses randomized backtracking in a complete boolean satisfiability solver which incorporates clause (nogood) learning to ensure completeness. Jussein & Lhomme [4] perform a constructive search while keeping conflict sets (nogoods) in a Tabu list and backtrack via a stochastic local search in the space of conflict sets.

Our method, called *Systematic Local Search* (SysLS) [3], follows the opposite approach. We incorporate systematicity within an inherently stochastic search method (like [2]). SysLS searches through a space of complete variable assignments and relaxes the requirement for maintaining feasibility. It preserves full freedom to move heuristically in the search space with maximum heuristic information available. While many local search methods easily get trapped in local optima, SysLS records local optima as nogoods in a search memory. Nogoods force the search away from these maximally consistent but unacceptable solutions. Our method is analogous to other diversification mechanisms in local search (*eg- Tabu search*) but is systematic and inherits the sound resolution rule for nogood learning. In this paper, we extend SysLS for optimization and, in particular, for job shop scheduling problems.

2 Systematic Local Search for Optimization

We begin this section with relevant definitions for the SysLS schema from [3]. A CSP is a tuple (V, D, C) where V is a set of variables with domains D and C is a set of k-ary constraints ($k \leq |V|$) on k variables in V.

Definition 1. *A nogood is a set of variable assignments* $\lambda_\perp = \{\langle x = a\rangle\}_{x \in X}, X \subseteq V$*, such that no solution to the CSP contains the variable assignments of* $\lambda_\perp$*.*

Definition 2. *Given a nogood search memory,* Γ*, an assignment* $\langle x = a\rangle$ *is* disallowed *if and only if* $\exists \lambda_\perp \in \Gamma, \langle x = a\rangle \in \lambda_\perp$ *and* $\forall \langle x' = a'\rangle \in \lambda_\perp \setminus \{\langle x = a\rangle\}$ *is the current assignment of the variable* x'*. Otherwise the assignment is* allowed*.*

Definition 3. *The* live domain *of a variable* x *is* $\Delta_x = \{a \in D_x | \langle x = a\rangle$ *is allowed*$\}$*.*

When the live domain of a variable is empty, *nogood resolution* allows the inference of a new nogood from the nogoods disallowing the domain elements. We define a *valuation function* $f_x(a)$ for a variable assignment $\langle x = a\rangle$ which is dependent on the current assignments of the other variables in V. The valuation function is used to guide the search towards feasible solutions. We classify each variable into one of four possible classes within the context of minimizing $f_x(a)$.

1. *MAXIMAL*: A variable x is *maximal* if and only if its current assignment $\langle x = a\rangle$ is such that $\forall b \in \Delta_x, f_x(a) \leq f_x(b)$.
2. *SUBMAXIMAL*: A variable x is *submaximal* if and only if $\exists b \in \Delta_x$ such that $f_x(b) < f_x(a)$.
3. *INF*: A variable x is in an *infeasible* state if its current assignment is $\langle x = a\rangle$ but $a \notin \Delta_x$.
4. *NULL*: Otherwise, x is currently not assigned a value.

In [3], the valuation function $f_x(a)$ for solving CSPs is the number of constraints violated when $\langle x = a\rangle$ given the other current variable assignments (eg-*min-conflicts*). Hence, SysLS perceives the CSP as an optimization problem to minimize the number of violated constraints while inducing new nogoods at every local minima. In this paper, we consider extensions to SysLS for constraint optimization problems (COPs). A solution to a COP is a complete assignment that both satisfies all the constraints and optimizes the objective function, C_{opt}. For most optimization problems, a problem-specific neighbourhood is available for moving in the space of complete feasible assignments. So, here we search in the space of *feasible* variable assignments and concentrate on improving the objective function. Hence, at every local optimum, the constraints are satisfied but the objective value might not be optimal. We wish to record a nogood that captures the subset of variable assignments responsible for the objective value, thus preventing non-solutions w.r.t. optimizing the objective function. Relying on a neighbourhood that preserves feasibility, we can consider the valuation function $f_x(a)$ that guides the search simply as the value of C_{opt} when x is reassigned to a. We can now rewrite the original SysLS algorithm for solving optimization problems. Figure 1 shows the new search method called $SysLS_{opt}$-NG. The algorithm receives as input of a set of variables V and the objective function C_{opt}. It outputs the first optimal solution or the best solution found. The variable ordering $select(V)$ is based on the classification of variables into the classes described above [1]. It is specified by a precedence order among the classes and then

[1] For the $SysLS_{opt}$-NG instance, we ignore the *NULL* class because we always maintain a complete assignment.

```
   Input: variable set V and C_opt
   Output: best solution found
 1 α ← initial complete assignment;
 2 repeat
 3 |   while not all x ∈ α are MAXIMAL do
 4 |   |   let x = select(V) ;
 5 |   |   assign(x) ;
 6 |   |   update the best solution if a better solution is found ;
 7 |   end
 8 |   λ⊥ = label(C_opt) ;
 9 |   add λ⊥ to the nogood cache Γ ;
10 |   simplify Γ (as in [2]) ;
11 |   if λφ ∈ Γ then return the best solution found
12 until stopping criterion;
13 return the best solution found
```

Fig. 1: $SysLS_{opt}$-NG finds an optimal solution for variables V with optimization constraint C_{opt}.

an order between the variables in each class. $SysLS_{opt}$-NG is parameterized by the variable ordering relation similarly to the original SysLS schema.

Once a variable is chosen, the action taken is specified by $assign(x)$ according to the following state transition rules for variables:

1. If the variable is in the *MAXIMAL* state, do nothing.
2. If the variable is in the *SUBMAXIMAL* state, then we reassign x to one of its maximal values.
3. If the variable is in the *INF* state, we switch the assignment to the maximal allowed domain element. If all elements are disallowed in the current solution, no action is taken until one of its assignments becomes *allowed*.

3 Case Study: the Job Shop Scheduling Problem

We provide a case study of applying $SysLS_{opt}$-NG to *job shop problems* (JSPs). Every JSP has n jobs and m machines. Each job $j \in J$ consists of exactly m operations and each of the m operations is processed on a different machine. The precedence relations between operations of the same job (R_J) are fixed. Each operation has a specified duration. For a complete schedule, no two operations can execute on the same machine at the same time. The objective function (C_{opt}) is to minimize the makespan, *i.e.* the time at which the last operation finishes execution. The JSP is known to be NP-hard.

Below we formulate the JSP as a constraint optimization problem. A variable v is defined as the ordering between two different operations processed on the same machine. For every variable, the domain D contains only two values $D = \{\prec, \succ\}$ specifying the ordering. The subset of $m \times (n-1)$ variables corresponding to the processing sequence on the m machines is denoted V^1. A

complete JSP schedule can be represented by a *directed acyclic graph* (DAG) G where the nodes are operations and precedence relations in R_J and V^1 are the arcs between operations whose length is equal to the duration of the source operation. Every operation is scheduled at its earliest start time determined by G and the makespan is equal to the length of the critical path in G. The critical path involves arcs corresponding to some precedence relations in R_J and some variables in V^1. Such a set of variables on the critical path is denoted by V^2 ($V^2 \subseteq V^1$). If the neighbourhood only changes variable assignments in V^2, then the schedule remains feasible as shown in [5]. At a local optimum, $SysLS_{opt}$-NG induces a nogood on the subset of variable assignments responsible for the makespan. Since it is solely determined by the critical path, the nogood contains only the variable assignments in V^2.

Theorem 1. *If any nogood recorded at a local optima represents the assignment of V^2, then nogood resolution is sound with respect to finding the optimal solution. (Proof omitted)*

The parameterized components of the algorithm (Fig 1) are implemented as:

1. **initial solution** - The operations on a machine are ordered by their precedence ordering in their respective jobs.
2. **select(V)** - The precedence between variable classes is $SUBMAXIMAL \prec INF$. The variable ordering within the variable class $SUBMAXIMAL$ is best-improvement. Under class INF, we choose a recency-based diversification scheme. The preference is first given to the oldest variable assignment. Ties are broken by the shorter makespan value. In addition, we never choose an INF variable whose live domain is empty.
3. **stopping criterion** - Stop when either known optimal makespan is reached or the maximum number of iterations has been executed.

4 Experimental Results

The empirical evaluation of $SysLS_{opt}$-NG is on the well-known Lawrence benchmark [2]. We compare our results with two standard heuristic approaches, simulated annealing (**SA**) [5] and Tabu search (**TB**) [1]. We also consider two state-of-the-art specialized approaches: a fast taboo search (**TSF**) [7], and a tabu search using shifting bottleneck procedure (**TSB**) [8]. The maximum number of iterations is set to 20000 similarly to the other authors. All the experiments are run on a PC with an Intel 2.8 GHz processor and 512 MB memory. Table 1 shows results in terms of the relative error [3]. Table 2 summarizes the best makespan, the mean relative error (MRE) and the mean run-time over 10 runs ($\bar{t}$) of some very hard instances. The $SysLS_{opt}$-NG performs extremely well on the 40 Lawrence instances where 27 are solved optimally (OPT in Table 1) and the best makespan values are generally within 2% of the optimal. The $SysLS_{opt}$-NG is

[2] from OR-Library at http://people.brunel.ac.uk/∼mastjjb/jeb/info.html

[3] $RE = \frac{(\mathcal{L}_{min} - \mathcal{L}_{opt})}{\mathcal{L}_{opt}} \times 100$ ($\mathcal{L}_{min}$ is the best makespan obtained and $\mathcal{L}_{opt}$ is the optimal makespan)

problem	$SysLS$	SA	TB	TSF	TSB
La01-05	0	0.30	0.51	0	0
La06-10	0	0	2.25	0	0
La11-15	0	0	1.43	0	0
La16-20	0.13	0.71	1.19	0	0
La21-25	0.64	1.23	3.29	0.10	0.10
La26-30	0.94	2.01	1.94	0.14	0.46
La31-35	0	0	0	0	0
La36-40	1.15	1.67	3.53	0.26	0.58
OPT	27	23	16	33	33
MRE	0.36	0.74	1.77	0.06	0.14

Table 1. Comparison of RE on Lawrence instances

Laxx	$SysLS$	$\bar{t}(sec)$	SA	TB	TSF	TSB
La19	843	11.5	848	860	842	842
La21	1055	308.1	1063	1099	1047	1046
La24	941	224.3	952	989	939	938
La25	979	234.7	992	995	977	979
La27	1255	810.8	1269	1258	1236	1235
La29	1177	872.8	1218	1206	1160	1168
La36	1275	455.8	1293	1302	1268	1268
La37	1411	743.0	1433	1453	1407	1411
La38	1212	634.4	1215	1254	1196	1201
La39	1249	639.5	1248	1269	1233	1240
La40	1240	652.5	1234	1261	1229	1233
MRE	1.14	–	2.13	3.63	0.23	0.52

Table 2. Comparison of best make span on a few hard Lawrence instances

a hybrid search method that combines desirable properties from systematic and stochastic search methods. The experimental results show that it finds better solutions than the general local search methods and compares favourably to the specialized heuristic approaches. Unlike other approaches which are specially designed for the JSP, the $SysLS_{opt}$-NG is a very general method for solving COPs.

References

1. Faruk Geyik and Ismail Hakki Cedimoglu. The strategies and parameters of tabu search for job-shop scheduling. *Journal of Intelligent Manufacturing*, 15(4):439–448, 2004.
2. M. L. Ginsberg and D. A. McAllester. GSAT and Dynamic Backtracking. In P. Torasso, J. Doyle, and E. Sandewall, editors, *The 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 226–237. Morgan Kaufmann, 1994.
3. William S. Havens and Bistra N. Dilkina. A Hybrid Schema for Systematic Local Search. In *Canadian Conference on Artifical Intelligence 2004*, 2004.
4. Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, July 2002.
5. Peter J. M. Van Laarhoven, Emile H. L. Aarts, and Jan Karel Lenstra. Job Shop Scheduling by Simulated Annealing. *Operations Research*, 50(1):113–125, 1992.
6. I. Lynce and J. Marques-Silva. Complete unrestricted backtracking algorithms for satisfiability. In *The 5th International Symposium on the Theory and Applications of Satisfiability Testing*, 2002.
7. Eugeniusz Nowicki and Czeslaw Smutnicki. A Fast Taboo Search Algorithm for the Job Shop Problem. *Management Science*, 42(6):797–813, 1996.
8. Ferdinando Pezzella and Emanuela Merelli. A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, 120:297–310, 2000.