New Modeling for the Total Tardiness Scheduling Problem using Constraint Programming

Marco Van Basten
Associazione Calcio Milan
Stadio Giuseppe Meazza, San Siro

Abstract

In this paper, a new modeling for the total tardiness single machine with setup-dependent scheduling problem is proposed using the constraint-based scheduling. We compare the performance of different algorithms using classical benchmark problems. The experiments show that algorithm efficiency improves with specific tree search strategies and heuristic scheduling variables.

1. Introduction

In recent years, constraint-based scheduling (CBS) has become a widely used form for modeling and solving scheduling problems using the constraint programming approach (Baptiste, LePape, and Nuijten 2001; Allahverdi et al. 2008). A scheduling problem is the process of allocating tasks to resources over time with the goal of optimizing one or more objectives (Pinedo 2002). A scheduling problem can be efficiently encoded like a constraint satisfaction problem (CSP). The activities, the resources and the constraints, which can be temporal or resource related, are the basis for modeling a scheduling problem in a CBS problem. Modeling objective function like the maximum completion time noted C_{max} in accordance with the notation of (Graham et al. 1979) is not hard. The situation becomes more difficult and complex for sum functions, such as the sum of the completion time $\Sigma(w_i)C_i$ or the total tardiness $\Sigma(w_i)T_i$ (Baptiste, LePape, and Nuijten 2001).

In this paper, a new modeling for the total tardiness ΣT_i is presented. The studied problem considers a single machine scheduling problem with sequence dependent setup times (SMSDST) with the objective of minimizing total tardiness of the jobs. This problem noted as $1|s_{ij}|\Sigma T_j$ may be defined as a set of *n* jobs available for processing at time zero on a continuously available machine. Each job *j* has a processing time p_j, a due date d_j, and a setup time s_{ij} which is incurred when job *j* immediately follows job *i*. It is assumed that all the processing times, due dates and setup times are non-negative integers. A sequence of the jobs *S* = $[q_0, q_1,..., q_{n-1}, q_n]$ is considered where q_j is the subscript of the j^{th} job in the sequence. The due date and the processing time of the j^{th} job in sequence are denoted as d_{q_j} and p_{q_j}, respectively. Thus, the completion time of the j^{th} job in sequence will be expressed as $C_{q_j} = \sum_{k=1}^{j}(s_{q_{k-1}q_k} + p_{q_k})$ while the tardiness of the j^{th} job in sequence will be expressed as $T_{q_j} = max(0, C_{q_j} - d_{q_j})$. The objective of the scheduling problem studied is to minimize the total tardiness of all the jobs which will be expressed as $\sum_{j=1}^{n} T_{q_j}$.

Different approaches have been proposed by a number of researchers to solve the $1|s_{ij}|\Sigma T_j$ problem. Rubin and Ragatz (1995) proposed a B&B, which quickly showed its limitations. It could solve to the optimality only small instances of benchmark files of 15, 25, 35 and 45 jobs proposed by the author. Bigras, Gamache, and Savard (2008) solved to the optimum all instances proposed by Rubin and Ragatz (1995) using a B&B approach with LP relaxation bounds. Many researchers used a wide variety of methods to solve this problem such as genetic algorithm (Gupta and Smith 2006; Sioud, Gravel, and Gagné 2009), ant colonies optimization (Gagné, Price, and Gravel 2002; Liao and Juan 2007) and Tabu/VNS (Gagné, Gravel, and Price 2005). Heuristics such as Random Start Pairwise Interchange (RSPI) (Rubin and Ragatz 1995) and Apparent Tardiness Cost with Setups (ATCS) (Lee, Bhaskaram, and Pinedo 1997) have also been proposed for solving such a problem. For their part, Kelbel and Hanzalek (2006) used a hybrid approach integrating constraint programming in a MIP algorithm to solve an industrial problem of lacquers. This problem is seen to be as a job shop scheduling problem. The main problem is divided into sub problems solved by the CBS approach. The MIP algorithm, then, solves the master problem. Hooker (2006) used the same approach as Kelbel and Hanzalek (2006) to solve an allocation problem. To our knowledge, there exists no other studies dealing with the solving of a total tardiness scheduling problem using the constraint programming approach.

The paper is organized as follows: the constraint-based scheduling is described in Section 2, while Section 3 deals with the modeling of the SMSDST and describes different approaches to the search procedure. Results of experiments are presented in Section 4. As a constraint programming environment we choose the ILOG IBM CP environment using ILOG Solver and Scheduler via the C++ API (ILO 2003b; 2003a).

2. The constraint-based scheduling

Constraint solving methods such as domain reduction and constraint propagation have proven to be well suited for a wide range of industrial applications (Fromherz 1999). These methods are increasingly combined with classical solving techniques from operations research, such as linear, integer, and mixed integer programming (Talbi 2002), to yield powerful tools for CBS by adopting them. The most significant advantage of using such CBS is to separate the model from the algorithms which solve the scheduling problem. This makes it possible to change the model without changing the algorithm used and vice-versa. Based on representations and techniques of constraint programming, various types of variables and constraints have been developed specifically for scheduling problems. Indeed, the domain variables may include interval domains where each value represents an interval (processing or early start time for example) and variable resources for many classes of resources. Similarly, various research techniques and constraints propagation have been adapted for this kind of problem. The following paragraph will show the various components involved in creating a model for a scheduling problem based on constraint programming.

The basic element is the activity in CBS. An activity represents the task or a job to be executed in a ressource. For any activity A_i, three variables are introduced : *start(A_i)*, *end(A_i)* and *proc(A_i)*, which respectively represent the start, end and the processing time of activity A_i. If r_i is the release date and d_i is the due date of activity A_i, then A_i must be executed in the interval [r_i, d_i]. Thus, the initial domain of *start(A_i)* and of *end(A_i)* are [r_i, lst_i] and [eet_i, d_i], respectively and where lst_i is the latest start time and eet_i represents the earliest end time. The execution time of an activity may be defined as the difference between the end and the start time of the activity : *proc(A_i) = start(A_i)- end(A_i)*. p_i represents the smallest value of field *proc(A_i)*. With this set of domains and variables, it is possible to encode a non-preemptive scheduling problem in a CSP. The temporal constraints between activities can be expressed by linear constraints between the start and the end of the activity variables. A precedence constraint between two activities A_i and A_j for example, can be expressed by *end(A_i) $\prec$ start(A_j)*. Such constraints can easily be propagated using a classical *arc-B-consistency* which is a special case of *arc-consistency* (Lhomme 1993).

In the context of the classical scheduling a resource is a machine. Constraints on resources represent the amount of resources used by the activities during the execution. Let A_i be an activity and *R* a resource with capacity *Cap(R)* and either *Cap(A_i, R)* is a variable representing the amount of the resource *R* required by the activity A_i. In order to represent a schedule, a variable *Cap(A_i, t, R)* is needed to represent the amount of the resource *R* required by A_i at a time *t*. Constraints on resources define and constrain the available resources by limiting the number of used resources. Resources can be renewable or consumable, and they may also be either unary or n-ary (LePape and Baptiste 1999). A setup or transition time between two activities A_1 and A_2 is defined by a setup time *Setup(A_1, A_2)* that must elapse between the A_1 end date time and the A_2 start time. In CBS, the setup time is related to the resource and the setup times are defined in a lattice which contains the setup times that must elapse between the A_i end date time activity in a resource R_m and the A_i start time activity in another resource $R_{m'}$. In some cases and configurations, it is possible to minimize the sum of the costs or transition time as an objective function.

To model objective function in CBS, a variable *obj* is added to the model and it will be minimized or maximized (LePape and Baptiste 1999). A constraint is then added to force this variable to be equal to the objective function. Several strategies can be used to minimize the value of *obj*. Thus, it is possible to iterate over the possible values contained in its domain, and this, ascending from a lower bound or descending from an upper bound. It is also possible to use a binary search in its domain. When the objective function is to optimize returns the makespan C_{max} or the maximum tardiness T_{max}, modeling and solving such problems is not difficult. The situation becomes more complex for objectives such as sum of (weighted) completion time, total (weighted) tardiness or the (weighted) number for tardy job (LePape and Baptiste 1999).

3. Modeling the SMSDST

In this section, we introduce the modeling of the machine and the activities used to solve the $1|s_{ij}|\Sigma T_j$ problem. Also, we introduce the modeling of the setup times and the objective function which, as a reminder, is the minimization of $\sum T_j$.

3.1 Modeling the Single Machine

To model a single machine with ILOG SCHEDULER 6.0 (ILO 2003a), we simply use the class IloUnaryResource. This allows handling unary resources, that is to say, resource whose capacity is equal to one. This resource can therefore not handle more than one job at a time. Thus, a constraint like disjunctive constraints (LePape and Baptiste 1999) is propagated during the search phase.

3.2 Modeling the Activities

To model the activities we use the class *IloActivity*. At the creation of each activity, we specify the execution time and the initial setup time which is introduced when this activity begins first in the sequence, and this, through method *IloActivity::setStartMin()*. We will, of course, add the constraint that links activities to the machine using the method *IloActivity::requires(IloUnaryResource)* which is a constraint on resources.

3.3 Modeling the Setup Times

The use of the setup or the transition times in CBS and also with ILOG SCHEDULER 6.0 (ILO 2003a) indicates that they are resource-related and not activity-related such as is the case in our problem. It is possible to overcome this problem by associating a type for each activity and creating transition time associated with these types. For this purpose, we perform four operations. First, we define a class *IloTransitionParam* which is managing and setting transition times. Then, we associate to this class the lattice of the setup times with the method *IloTransitionParam::setValue()*. After that, we add the *IloTransitionParam* class to the unary resource with an *IloTransitionTime* class. Thus, lattice of the setup times is associated with the unary machine. Finally, we assign each activity type with the method *IloActivity::setType()*. To simplify the formulation, activity A_i is of type *i*. As we have *n* activities, we have *n* types. Thus, when calculating the objective function, it is possible to associate the transition time between two distinct types of activities.

3.4 Modeling the Objective Function

To model the total tardiness, we must first define an *IloNumVar* variable *Tard*. Then we define an *IloNumVarArray* array *T* containing the tardiness T_i of the different activities A_i during the research phase. When we create the activities in the model, we add a constraint that combines the activities A_i to the corresponding T_i. We use method *IloActivity::getEndExpr()* which returns the end date of activity during the research phase. After that, we add a constraint which combines the variable *Tard* with the sum of the T_i in the table *T* with the method *IloSum (IloNumVarArray)*. Finally, we add a constraint that minimizes the variables using the method *IloMinimize (IloNumVar)*. Thus, we obtain the objective function which is added to the model with the method *IloModel::add ()*.

4. Experimental results and discussion

The benchmark problem set consists of eight instances each with a number of jobs of 15, 25, 35 and 45 jobs, and it is taken from the work of Ragatz (1993). These instances are available on the internet at https://www.msu.edu/ rubin/data/c&ordata.zip. The job processing times are normally distributed with a mean of 100 time units and the setup times are also uniformly distributed with a mean of 9.5 time units. Each instance has three factors which have both high and low levels. These factors are due date range, processing time variance and tardiness factor. The tardiness factor determines the expected proportion of jobs that will be tardy in a random sequence. All the experiments were run on an Itanium with 1.4 GHz processor and 4 GB RAM. To solve the $1|s_{ij}|\Sigma T_j$ problem, we use the ILOG Solver 6.0 (ILO 2003b) and ILOG Scheduler 6.0 (ILO 2003a).

As mentioned previously, the algorithm used for solving the problem can be developed independently of the formulation. And we must define what is called a goal to implement a search algorithm. ILOG Scheduler 6.0 (ILO 2003a) provides several predefined goals to model search algorithms. There are, indeed, six goals to create a predefined scheduling: *IloRankBackward*, *IloRankForward*, *IloSequenceBackward*, *IloSequenceForward*, *IloSetTimesForward* and *IloSetTimesBackward*. Only the last two goals may be used, because the first four goals are only used for scheduling resources and not activities on resources. Thus, we can use *IloSetTimesForward* which assigns values to variables representing the start time of all activities processed or *IloSetTimesBackward* which assigns values to variables representing the execution end time of all activities. For this last goal, we must define a horizon of execution for a sequence of activities, however. A horizon implementation defines the end date of execution before all activities. It thereby provides an upper bound to the execution end time of the various activities. For this we use the class *IloSchedulerEnv* which groups all the default settings for modeling a scheduling problem. Then, we call the method *IloSchedulerEnv: setHorizon (IloInt Horizon)*, which sets the horizon calculated previously. For this problem, we use the EDD rules (Pinedo 2002) to calculate the horizon. When using either of the goals *IloSetTimesForward* or *IloSetTimesBackward*, it is possible to define the heuristic scheduling variables representing start times (for *IloSetTimesForward*) or end times (for *IloSetTimesBackward*). These heuristics are called *activity selector*. ILOG Scheduler 6.0 (ILO 2003a) also offers four selectors activities. The first, *IloSelFirstActMinEndMin* returns the first activity that has the smallest start and end time. The second, *IloSelFirstActMinEndMax* returns the first activity that has the smallest start time and the greatest end time. The third *IloSelLastActMaxStartMax* returns the last activity that has the greatest start and end time. Finally, *IloSelLastActMaxStartMin* returns the last activity which has the greatest start time and the smallest end time. Thus, it is possible to use eight search algorithms: 2 goals with 4 activities selectors.

Table 1 summarizes the comparison between the results obtained using the eight algorithms: *IloSetTimesForward* with *IloSelFirstActMinEndMax* indexed by *FFMinMax*, *IloSetTimesForward* with *IloSelFirstActMinEndMin* indexed by *FFMinMin*, *IloSetTimesForward* with *IloSelLastActMaxStartMin* indexed by *FLMaxMin*, *IloSetTimesForward* with *IloSelLastActMaxStartMax* indexed by *FLMaxMax*, *IloSetTimesBackward* with *IloSelFirstActMinEndMax* indexed by *BFMinMax*, *IloSetTimesBackward* with *IloSelFirstActMinEndMin* indexed by *BFMinMin*, *IloSetTimesBackward* with *IloSelLastActMaxStartin* indexed by *BLMaxMin*, *IloSetTimesBackward* with *IloSelLastActMaxStartMax* indexed by *BLMaxMax*. The execution time is limited to 10 minutes. The first column of Table 1 refers to the name of the instance. The best results found by the different algorithms are marked in Table1. It is worth noting that the results of the algorithms *FFMinMax*, *FFMinMin*, *BLMaxMin* and *BLMaxMAx* give better results than the others algorithms. It can be explained by the fact that *FFMinMax* and *FFMinMin* return an activity which has the smallest setup time and sequence it first while *BLMaxMin* and *BLMAxMax* return an activity that has the greatest end date and sequence it last. Also, *FFMinMin* gives the best results with 22 times because this algorithm sequences first the activities with the minimum end date. To explore the search tree, ILOG Solver 6.0 (ILO 2003b) uses the default Depth-First Search (DFS) strategy. This method explores the tree in depth research from left to right. It selects,every time, the left leaf not yet visited. ILOG Solver 6.0 also provides three other strategies to explore the search tree : Slice-Based Search (SBS) (Beck and Perron 2000), Depth-Bounded Discrepancy Search (DDS) (Walsh 1997) and Interleaved Depth-First Search (IDFS) (Meseguer 1997).

Table 2 presents the results obtained by the application

PROB	*FFMinMax*	*FFMinMin*	*FLMaxMin*	*FLMaxMax*	*BFMinMax*	*BFMinMin*	*BLMaxMin*	*BLMaxMax*
401	90*	90*	885	622	656	656	90*	90*
402	0*	0*	2586	1242	2199	2273	0*	0*
403	3748	3633	5699	4503	4969	4857	3846*	3846*
404	1067*	1067*	5278	4288	4241	4249	1227	1227
405	0*	0*	184	186	184	95	0*	0*
406	0*	0*	1979	869	2852	2836	0*	0*
407	2126	2126	3889	2719	2870	3676	2110*	2110*
408	5915*	5915*	10511	7539	8456	7032	6077	6749
501	835	504	2085	1325	2085	1874	312*	560
502	0*	0*	3773	1575	1476	3778	0*	0*
503	5590	4179*	6924	5519	6542	5871	4490	4820
504	8468	0*	10078	10596	10324	10152	0*	0*
505	0*	0*	1720	790	886	1135	0*	0*
506	0*	0*	5178	4633	5431	5245	0*	0*
507	8868	8291*	16261	9254	10324	10985	10862	11051
508	6978	5383	12120	8010	9090	11235	2600*	2600*
601	368	34	5139	1885	1543	1290	40	32*
602	0*	0*	10690	7968	8876	10604	0*	0*
603	21756	20865*	31342	24366	25788	25110	25276	26926
604	39330	33841	40267	36827	34541	34328	32721*	33778
605	1534	1087*	6488	3061	3127	3637	734	1961
606	6580	0*	13092	7295	11236	9765	0*	0*
607	18746*	18952	30544	18519	23675	25437	22485	22164
608	22314	21426	33447	19137	24576	23993	10889*	11070
701	1353	163*	11734	3993	8976	5357	569	1233
702	5919	0*	19032	9738	9013	25345	0*	0*
703	34458	31875*	49659	40553	45609	48765	39600	39252
704	45157	43709	61741	48087	54329	47951	43513	40570*
705	725	333*	9284	5438	5430	4917	1123	3560
706	17854	4674	22865	9575	15432	14649	0*	0*
707	34272	32960*	52930	35090	42651	43994	36429	41783
708	55051	49373*	75009	42380	54079	55438	55464	59737

Table 1: Comparison of different algorithms

PROB	*DFS*	*DDS*	*SBS*	*IDFS*	*PROB*	*DFS*	*DDS*	*SBS*	*IDFS*
401	90*	90*	90*	90*	**601**	34*	137	34*	34*
402	0*	0*	0*	0*	**602**	0*	0*	0*	0*
403	3633	3418*	3418*	3443	**603**	20865	18729	18692*	18971
404	1067*	1067*	1067*	1067*	**604**	33841	24119*	25249	25666
405	0*	0*	0*	0*	**605**	1087	680	634	605*
406	0*	0*	0*	0*	**606**	0*	924	0*	449
407	2126	1861*	1861*	1898	**607**	18952	15259*	15291	15488
408	5915	5736	5712*	5773	**608**	21426	12646	13557	1202*1
501	504	446	345*	430	**701**	163	141	137	131*
502	0*	0*	0*	0*	**702**	0*	0*	0*	0*
503	4179	3877*	3885	4098	**703**	31875	28713*	29035	29577
504	0*	1254	1150	803	**704**	43709	30747	31977	30736
505	0*	0*	0*	0*	**705**	333	532	316*	316*
506	0*	0*	0*	0*	**706**	4674	2459	1327*	2803
507	8291	7598*	7627	7716	**707**	32960	28633	28172*	28811
508	5383	3431	3488	3407*	**708**	49373	37723	38112	36998*

Table 2: Comparison of different tree exploration strategies for the *FFMinMin* algorithm

PROB	*DDS*	*SBS*
401	1.26*	2.1
402	0.19	0.16*
403	332.33	244.68*
404	181.24	137.7*
405	0.07	0.02*
406	0.11	0.08*
407	38.32*	128.9

Table 3: Comparison of time execution

of the *FFMinMin* algorithm with the four strategies. The best results found by the different algorithms are marked. It will be observed from Table 2 that the SBS and DDS strategies provide noticeable results in particular. The SBS strategy gives 21 times the best results and the DDS 18 times. These algorithms could solve 13 of the 32 instances to optimality within 10 minutes of running time. Also, these two strategies solve to optimality 7 of the 8 small instances. The computation time of the SBS and DDS exploration tree strategies for the 7 instances solved to the optimality are shown in Table 3. SBS solve to the optimality 5 instances faster than DDS.

5. Conclusion

In this paper, a modeling for the single machine total tardiness problem with sequence-dependent setup times is proposed. The ILOG IBM CP environment is used to model this problem via the C++ API. After a brief introduction of constraint-based scheduling, we describe the modeling of the single machine, the activities, the setup times and the objective function.

From the experimental results, it may be concluded that the *IloSetTimesForward* algorithm with the *IloSelFirstAct-MinEndMin* activity selector gives the best results. Also, the DDS and the SBS exploration tree strategies give better results than the DFS strategy. The proposed algorithms solve to the optimality seven of the eight small instances in less than 10 minutes. One of the main reasons for concluding that CBS is highly efficient is its flexibility in modeling, which allows the user to model in a more natural way, and also separates the model from the algorithms that solve the scheduling problem. This allows testing of different algorithms and models.

For relatively larger problems, an implicit enumeration technique such as a branch-and bound algorithm may be developed. For even larger problems, a heuristic can be constructed. Finally, more research can be done on the propagation of the total tardiness, as well as on other tree search strategies and heuristic scheduling variables.

References

Allahverdi, A.; Ng, C.; Cheng, T.; and Kovalyov, M. Y. 2008. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research* 187(3):985 – 1032.

Baptiste, P.; LePape, C.; and Nuijten, W. 2001. *Constraint-Based Scheduling : Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers.

Beck, J. C., and Perron, L. 2000. Discrepancy bounded depth first search. In *CP-AI-OR'2000: Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 7–17.

Bigras, L.; Gamache, M.; and Savard, G. 2008. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization* 5(4):663–762.

Fromherz, M. P. 1999. Model-based configuration of machine control software. Technical report, In Configuration Papers from the AAAI Workshop.

Gagné, C.; Gravel, M.; and Price, W. L. 2005. Using metaheuristic compromise programming for the solution of multiple objective scheduling problems. *The Journal of the Operational Research Society* 56:687–698.

Gagné, C.; Price, W.; and Gravel, M. 2002. Comparing an aco algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. *Journal of the Operational Research Society* 53:895–906.

Graham, R. L.; Lawler, E. L.; Lenstra, J. K.; and Rinnooy Kan, A. G. H. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 5:287–326.

Gupta, S. R., and Smith, J. S. 2006. Algorithms for single machine total tardiness scheduling with sequence dependent setups. *European Journal of Operational Research* 175(2):722–739.

Hooker, J. N. 2006. An integrated method for planning and scheduling to minimize tardiness. *Constraints* 11(2-3):139–157.

ILOG. 2003a. *ILOG Scheduler 6.0. User Manual.*

ILOG. 2003b. *ILOG Solver 6.0. User Manual.*

Kelbel, J., and Hanzalek, Z. 2006. A case study on earliness/tardiness scheduling by constraint programming. In *Proceedings of the CP 2006 Doctoral Programme*, 108 – 113.

Lee, Y.; Bhaskaram, K.; and Pinedo, M. 1997. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions* 29:45–52.

LePape, C., and Baptiste, P. 1999. Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem. *Journal of Heuristics* 5(3):305–325.

Lhomme, O. 1993. Consistency techniques for numeric csps. In *IJCAI'93: Proceedings of the 13th international joint conference on Artifical intelligence*, 232–238. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Liao, C., and Juan, H. 2007. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. *Computers and Operations Research* 34:1899–1909.

Meseguer, P. 1997. Interleaved depth-first search. In *IJ-CAI'97: Proceedings of the Fifteenth international joint conference on Artifical intelligence*, 1382–1387. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Pinedo, M. 2002. *Scheduling Theory, Algorithm and Systems*. Prentice-Hall.

Ragatz, G. L. 1993. A branch-and-bound method for minimumtardiness sequencing on a single processor with sequence dependent setup times. In *Proceedings twenty-fourth annual meeting of the Decision Sciences Institute*, 1375–1377.

Rubin, P., and Ragatz, G. 1995. Scheduling in a sequence-dependent setup environment with genetic search. *Computers and Operations Research* 22:85–99.

Sioud, A.; Gravel, M.; and Gagné, C. 2009. New crossover operator for the single machine scheduling problem with sequence-dependent setup times. In *GEM'09: The 2009 International Conference on Genetic and Evolutionary Methods*.

Talbi, E.-G. 2002. A taxonomy of hybrid metaheuristics. *Journal of Heuristics* 8:541–564.

Walsh, T. 1997. Depth-bounded discrepancy search. In *IJCAI'97: Proceedings of the Fifteenth international joint conference on Artifical intelligence*, 1388–1393. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.