Leap Before You Look: An Effective Strategy in an Oversubscribed Scheduling Problem*

Laura Barbulescu and **L. Darrell Whitley** and **Adele E. Howe**
Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
email:{laura,whitley,howe}@cs.colostate.edu

Abstract

Oversubscribed scheduling problems require removing or partially satisfying tasks when enough resources are not available. For a particular oversubscribed problem, Air Force Satellite Control Network scheduling, we find that the best approaches make long leaps in the search space. We find this is in part due to large plateaus in the search space. Algorithms moving only one task at a time are impractical. Both a genetic algorithm and Squeaky Wheel Optimization (SWO) make long leaps in the search space and produce good solutions almost 100 times faster than local search. Greedy initialization is shown to be critical to good performance, but is not as important as directed leaps. When using fewer than 2000 evaluations, SWO shows superior performance; with 8000 evaluations, a genetic algorithm using a population seeded with greedy solutions further improves on the SWO results.

Introduction

Variants of local search have been shown to be extremely effective and robust for scheduling problems (Vaessens, Aarts, & Lenstra 1996; Lemaître, Verfaillie, & Jouhaud 2000). When heuristics are available for focusing attention on parts of the schedule, then iterative repair performs well (e.g., (Kramer & Smith 2003)). Both these classes of algorithms adopt an incremental view: propose a change, evaluate the result, accept or retract the change, and identify the next potential changes. The incremental strategy works well because minor changes may cause a cascade effect in the schedule and these cascade effects are difficult to predict.

In contrast, a genetic algorithm, *Genitor* (Whitley 1989), has shown better performance on an oversubscribed scheduling application: Air Force Satellite Control Network (AFSCN) access scheduling (Barbulescu *et al.* 2004b). *Genitor* proposes large changes to a schedule at each iteration. It represents a proposed schedule as a permutation (a prioritization) of the tasks, which can be translated into an actual schedule with assignment of tasks to time slots on resources. The crossover operator (Syswerda 1991) randomly selects about half of the positions in a permutation for re-ordering, typically producing a large change in the schedule.

One hypothesis about the superior performance is that the genetic algorithm is *learning* patterns of task interactions. An analysis of the search space and solutions showed that a simple domain specific pattern emerged and provided evidence for more complex patterns (Barbulescu *et al.* 2004a).

Aside from such patterns, another explanatory hypothesis is that the large leaps taken by *Genitor* at each iteration are needed to effectively traverse the search space. We find evidence to support this hypothesis in the performance of another method that makes directed leaps: Squeaky Wheel Optimization (SWO)(Joslin & Clements 1999). SWO focuses attention on the most contentious tasks in the schedule and simultaneously moves those requests forward in the prioritized permutation, thereby causing other less troublesome tasks to move backward. We show that solutions appear to be residing on plateaus, which tend to be difficult for incremental methods to traverse. We also examine whether the power is obtained from simply starting closer to the best solutions (i.e., the role of initialization) and whether *Genitor* and SWO seem to be following similar paths to their solutions. We find that narrowing the distance to the optimal solutions certainly helps, but is not the only factor at work, and that *Genitor* and SWO follow rather different paths through the search space. Thus, these algorithms represent two different, but almost equally effective methods for making large leaps across the search space.

AFSCN Scheduling

The U.S.A. AFSCN is responsible for coordinating communications between users on the ground and satellites in space. Communications to more than 100 satellites are performed through 16 antennas at nine ground stations located around the globe. To reserve a particular antenna for a period of time, users submit a task request which includes a required duration, a time window within which the duration must occur and a desired resource. Alternate time windows and antennas may also be specified. Approximately 500 requests are typically received for a single day. Separate schedules are produced for each day.

Some antennas are typically oversubscribed. After human schedulers attempt to fit all tasks into the schedule, often about 120 conflicts, or requests that could not be accommo-

*With apologies to Keith Golden, we have co-opted the clever title to show how it applies to scheduling as well as planning dated, remain. Because satellites are extremely expensive resources and the tasks may be mission critical, absolute rejection of requests is not an option. Rather, human schedulers must engage in a complex, time-consuming arbitration process to create a conflict-free schedule.

For our studies, we use 12 days of actual data from the AFSCN: seven consecutive days from 1992 and five days from 2002-2003[1]. These data demonstrate the changing nature of the task (e.g., differences in size and differences in utilization of resources between 1992 and 2002/2003) and exhibit the complex structure of real problems, which we have not been able to replicate in synthesized problems.

Analyses of these problems have shown a high degree of contention for resources, which leads to a high branching factor for most search algorithms (Barbulescu *et al.* 2004a). *Genitor* has shown the best performance on this problem so far (Barbulescu *et al.* 2004b; 2004a), finding schedules that minimize the number of conflicts. It appears that *Genitor* may be learning patterns, some that are easily translated into heuristics (e.g., schedule the low altitude satellite requests first) and some that are more complex.

The primary criterion that has been used in attempts to automate AFSCN scheduling is minimizing the number of conflicts (e.g., (Gooley 1993; Parish 1994)). This results in a very coarse evaluation function for an NP-hard problem where the search space is exponentially large (Barbulescu *et al.* 2004b). Minimizing conflicts also means that large tasks are generally bumped more often than smaller tasks. Conversations with human schedulers indicate that they often negotiate changes in *duration* of tasks to fit in additional tasks–and that everything must eventually be scheduled, even if tasks must be modified. Thus, a better evaluation criterion and the one used in this paper is to minimize the sum of overlaps between conflicting tasks in a proposed schedule. This provides the human schedulers with a better starting point for negotiations. It also provides a richer evaluation function.

Related Work

Other examples of multiple resource, oversubscribed problems include applications such as USAF Air Mobility Command (AMC) scheduling (Kramer & Smith 2003), NASA's shuttle ground processing (Deale *et al.* 1994), and satellite scheduling (Frank *et al.* 2001; Globus *et al.* 2003).

AMC scheduling assigns delivery missions to air wings. Kramer and Smith (Kramer & Smith 2003) adopt an iterative repair approach by greedily creating an initial schedule by priority order and then attempting to insert unscheduled tasks by retracting and re-arranging conflicting tasks.

Globus et al.(2003) compared a genetic algorithm, simulated annealing, SWO and hill climbing on a simplified, synthetic form of the satellite scheduling problem (two satellites with a single instrument). The genetic algorithm and SWO techniques were reported to perform poorly. However, similarly to what we found for AFSCN scheduling, multiple moves performed better than single moves.

For both AFSCN scheduling and MAXSAT, it is possible to quickly identify key variables or tasks that are unsatisfied. In both cases, there is an opportunity to resolve these unsatisfied variables or requests and to use a greedy selection strategy for changing a variable or schedule assignment.

Another similarity is that the search spaces in both cases have large flat regions or plateaus (Gent & Walsh 1993). In MAXSAT, these plateaus result from the fact that the evaluation function is a linear combination of subfunctions: each clause is a subfunction. In oversubscribed scheduling problems, moves affecting only requests that do not compete for the same resources or time windows have no impact on the evaluation function. Strategies for "walking" across these flat regions are therefore key components for successful search in both domains. One strategy that is similar in flavor to the SWO strategy used here is JumpSAT (Gent & Walsh 1995). Instead of flipping *one* variable in an unsatisfied clause, one variable is flipped in *each* unsatisfied clause. While not state of the art for MAXSAT, JumpSat was shown to outperform greedy local search (the GSAT algorithm).

MAXSAT and AFSCN differ in two important ways. First, the neighborhood size is $O(N)$ for MAXSAT while it is $O(N^2)$ for scheduling (where N is the number of variables or requests). Second, MAXSAT has a fast partial evaluation that allows one to quickly evaluate a move. Our scheduling domain requires that a new schedule be built from scratch and evaluated after every move.

Minimizing Overlap in AFSCN

A wide range of approaches have been tested for minimizing conflicts in AFSCN scheduling. One of the earliest, Gooley (1993), exploited complex heuristics and mixed integer programming. Attempts using constraint based methods (e.g., texture-based (Beck *et al.* 1997) and minslack (Smith & Cheng 1993)) worked well on small, synthetic problems, but did not scale well to the actual data (Barbulescu *et al.* 2002); similarly, attempts to use constructive tree search algorithms, such as HBSS (Bresina 1996), have so far not produced good results, probably because the ordering heuristics are inadequate.

Some of the approaches do not translate well for minimizing overlaps because they were carefully tuned for the original evaluation metric. Local search and *Genitor* are easily adapted. Additionally, because the evaluation metric is less coarse, new algorithms such as SWO are viable. This section describes the set of algorithms and their performance.

Algorithms

The three algorithms considered encode solutions using a permutation π of the n task request IDs (i.e., $[1..n]$); a *schedule builder* is used to generate solutions from the permutation. In the order defined by π, each task request is assigned to the first available resource from its list of alternatives and at the earliest possible starting time. If a request cannot be scheduled without conflict on any of the alternative resources, it overlaps; we assign such a request to the alterna-

[1]We thank Dr. James T. Moore, Associate Professor, Dept. of Operational Sciences, Air Force Institute of Technology and Brian Bayless and William Szary from Schriever Air Force Base for providing the data.

tive resource on which the overlap with requests scheduled so far is minimized.

Genetic algorithms were found to perform well in some early studies (Parish 1994) and for an abstraction of NASA's Earth Observing Satellite (EOS) scheduling problem (Wolfe & Sorensen 2000). For our studies, we use the version of *Genitor* originally developed for a warehouse scheduling application (Starkweather *et al.* 1991). Like all genetic algorithms, *Genitor* maintains a population of solutions. In each step of the algorithm, a pair of parent solutions is selected, and a crossover operator is used to generate a single child solution, which then replaces the worst solution in the population. Selection of parent solutions is based on the rank of their fitness, relative to other solutions in the population. Following Parish (1994), we use Syswerda's (1991) position-based crossover operator.

As the local search algorithm, we implemented a hill-climber. Because it has been successfully applied to a number of well-known scheduling problems, we selected a domain-independent move operator, the *shift* operator. From a current solution π, a neighborhood is defined by considering all $(N-1)^2$ pairs (x, y) of positions in π, subject to the restriction that $y \neq x-1$. The neighbor $\pi^{'}$ corresponding to the position pair (x, y) is produced by *shifting* the job at position x into the position y, while leaving all other relative job orders unchanged. Given the large neighborhood size, we use the shift operator in conjunction with next-descent hill-climbing: the neighbors of the current solution are examined in a random order, and the first neighbor with either a lower or equal sum of overlaps is accepted.

SWO (Joslin & Clements 1999) repeatedly iterates through a cycle composed of three phases. First, a greedy solution is built, based on priorities associated with the elements in the problem. Then, the solution is analyzed and the elements causing "trouble" are identified, based on their contribution to the objective function. Third, the priorities of such "trouble makers" are modified, such that they will be considered earlier during the next iteration. The cycle is then repeated, until a termination condition is met.

In our implementation of SWO, we identify the overlapping requests as the "trouble spots" in the schedule. We sort the overlapping requests in increasing order of their contribution to the sum of overlaps. We associate with each such request a distance to move forward, based on its rank in the sorted order. We fix the minimum distance of moving forward to one and the maximum distance to five (this seems to work better than other possible values we tried). The distance values are equally distributed among the ranks. We move the requests forward in the permutation in increasing order of their contribution to the sum of overlaps (smaller overlaps first). We tried versions of SWO where the distance to move forward is proportional with the contribution to the sum of overlaps or is fixed. However, these versions performed worse than the rank based distance implementation described above.

We construct the initial greedy permutation for SWO by sorting the requests in increasing order of their flexibility. Our flexibility measure is identical to that defined for AMC (Kramer & Smith 2003): the duration of the request divided by the average time window on the possible alternative resources. We break ties based on the number of alternative resources available. For requests with equal flexibilities and numbers of alternative resources, the earlier request is scheduled first. For multiple runs of SWO, we restart it from a modified permutation created by performing 20 random swaps in the initial greedy permutation.

Performance

The results of running each of the algorithms are summarized in Table 1. For each of the three algorithms, we report the best and mean value and the standard deviation observed over 30 runs, with 8000 evaluations per run. CPU times corresponding to 30 runs of each algorithm (on a Dell Precision 650, 3.06 GHz Xeon, running Linux) are also included. "Size" is the number of requests for that date. In the third column, we include the best values we have seen for the sum of overlaps; these values were obtained by running *Genitor* with the population size increased to 400 and up to 50,000 evaluations. Both *Genitor* and local search were initialized from random permutations.

While *Genitor* and SWO perform equally well for the data from 1992 and R5, SWO produces the best results for the remaining data. The performance of local search is relatively poor: worse mins/means and higher variance. If SWO is allowed up to 50,000 evaluations, SWO does not find better solutions, unlike *Genitor* which continues to improve the solution quality.

Leap or Creep?

Both SWO and *Genitor* make large changes to the permutation at each step before evaluating the resulting schedule. Thus, another hypothesis for *Genitor*'s success is that effectively searching the space requires a large step size or "leaping before looking". Local search adopts a conservative approach by assessing the results of each change. Local search can find close to optimal solutions for these problems provided that it searches for a very long time (500,000 evaluations). In this case, the search space may be too big, and the information gained at each evaluation may be too little to justify the expense of evaluation after each move.

To test the hypothesis that large step size is critical to good performance, we look at four factors. First, we examine the neighborhood of permutations to assess whether the search space does contain plateaus and more specifically, whether solutions appear to reside on them. Second, we assess the role of initialization in traversing the space: are Genitor and SWO succeeding because they just start out closer to the solution? Third, we control for the effects of multiple moves by comparing SWO to a version that makes one move at a time. Finally, we look at the rate of solution improvement over time for both SWO and *Genitor*.

Plateaus

One reason local search has trouble with AFSCN scheduling are plateaus. We count the number of pairwise shifts in 30 random and 30 best known solutions that produce no

Day	Date	Size	Best Known	*Genitor*				Local Search				SWO			
				Min	Mean	Stdev	CPU	Min	Mean	Stdev	CPU	Min	Mean	Stdev	CPU
A1	10/12/92	322	104	104	106.9	0.6	107	255	375.0	54.4	96	104	104	0.0	97
A2	10/13/92	302	13	13	13	0.0	101	65	174.6	50.6	92	13	13.4	2.0	91
A3	10/14/92	311	28	28	28.4	1.2	102	144	252.0	52.9	91	28	28.1	0.6	89
A4	10/15/92	318	9	9	9.2	0.7	107	153	239.6	55.4	97	9	13.3	7.8	95
A5	10/16/92	305	30	30	30.4	0.5	104	142	220.1	59.3	93	30	30	0.0	91
A6	10/17/92	299	45	45	45.1	0.4	105	190	277.4	46.7	93	45	45.1	0.3	92
A7	10/18/92	297	46	46	46.1	0.6	90	137	219.6	40.4	81	46	46	0.0	79
R1	03/07/02	483	774	913	987.8	40.8	189	1559	1830.9	143.4	175	798	841.4	14.0	178
R2	03/20/02	457	486	519	540.7	13.3	162	1078	1235.5	92.8	146	491	503.8	6.5	148
R3	03/26/03	426	250	275	292.3	10.9	144	788	967.7	96.7	133	265	270.1	2.8	130
R4	04/02/03	431	725	738	755.4	10.3	140	1139	1287.1	84.2	126	731	736.2	3.0	125
R5	05/02/03	419	146	146	146.5	1.9	127	351	457.9	69.1	112	146	146.0	0.0	109

Table 1: Performance of *Genitor*, local search and SWO in terms of the best and mean sum of overlaps. All statistics are taken over 30 independent runs, with 8000 evaluations per run. Best values we have obtained for these problems are included in the third column. Dates are given shorthand names to indicate whether they belong to the AFIT set (A) or are recent (R). CPU times (in seconds) corresponding to the 30 runs of each algorithm are also shown.

change in the resulting schedule (non-interacting pairs). Table 2 shows the mean number of non-interacting pairs of tasks. *Average Percentage* corrects for problem size differences. More than 33% of the shifts make no change to the schedule. Best known solutions (except for R3) have slightly more non-interacting pairs than do random solutions.

Day	Total Pairs	Non-interacting Pairs			
		Random Perms		Optimal Perms	
		Mean	Avg %	Mean	Avg %
A1	51681	20767.2	40.2	22925.8	44.3
A2	45451	19684.3	43.3	21169.7	46.6
A3	48205	19859.6	41.2	20373.5	42.3
A4	50403	19491.7	38.7	20729.1	41.1
A5	46360	18371.3	39.6	19238.9	41.5
A6	44551	16771.7	37.6	17811.7	39.9
A7	43956	18406.9	41.9	20220.7	46.0
R1	116403	39380.1	33.8	37805.9	36.2
R2	104196	34362.7	32.9	33894.3	37.4
R3	90525	33470.8	36.9	41285.6	35.5
R4	92665	34147.1	36.8	36816.9	39.7
R5	87571	34180.8	39.0	35891.4	40.9

Table 2: Mean and average % for the number of pairs of non-interacting requests over 30 random and optimal permutations. Second column is the total number of request pairs in the permutation.

Intuition might suggest that *Genitor* is likely to locate and jump onto the middle of a large plateau, while gradient methods might be more likely to find the edge of a plateau. To test this idea, we apply a random walk to solutions from Genitor and local search with the constraint that after each shift the new permutation has the same value as the initial one. We then compute the pairwise distance after increments of shifts (10, 50, 100 and 500). *Genitor*'s solutions drift somewhat further than do those of local search (average distance of 19869 versus 19350). However, the difference is small, and neither ever revisits the same point in the space even after 500 shifts.

Day	*Seeded Genitor*			*GreedyStartLS*		
	Min	Mean	Stdev	Min	Mean	Stdev
A1	104	107.9	3.0	104	122.5	10.6
A2	13	13	0.0	13	23.4	14.9
A3	28	28	0.0	28	33.0	6.1
A4	9	9	0.0	9	14.4	10.1
A5	30	30	0.0	30	43.1	13.8
A6	45	45	0.0	45	47.5	5.5
A7	46	46	0.0	46	81.6	22.9
R1	794	828.3	19.3	958	1062.4	41.4
R2	486	494.5	7.7	554	600.7	26.2
R3	250	257	5.9	356	451.8	37.9
R4	725	730.6	6.5	754	848.5	51.6
R5	146	146	0.0	146	153.4	8.1

Table 3: Performance of Genitor and local search when initialized from greedy permutations. All statistics are taken over 30 independent runs, with 8000 evaluations per run.

Role of Initialization

While *Genitor* and local search normally start with random permutations, SWO is initialized with a greedy solution. The greedy solution can be quite good; for two problems (A6 and R5), the greedy solution translated into best known values. In this section, we investigate the effects of initializing the search from similar greedy solutions for all three algorithms considered.

We modify the local search algorithm by starting it from the SWO greedy initial permutation and from variations of it obtained by randomly swapping 20 pairs of tasks (as in the initialization for SWO); we call this *GreedyStartLS*. We also seed the initial population of *Genitor* (size 200) with the greedy initial permutation built for SWO and 199 variations of this permutation, obtained by randomly swapping 20

Day	Size	SWO1move		
		Min	Mean	Stdev
A1	322	113	115.0	4.2
A2	302	13	13.1	0.3
A3	311	28	29.2	1.9
A4	318	9	11.6	3.0
A5	305	30	31.8	2.7
A6	299	45	45.9	1.7
A7	297	46	49.1	3.7
R1	483	999	1125.2	54.6
R2	457	600	627.6	13.3
R3	426	288	313.7	15.8
R4	431	820	842.0	10.9
R5	419	146	157.0	7.9

Table 4: Performance of a modified version of SWO where only one request is moved forward. All statistics are taken over 30 independent runs, with 8000 evaluations per run.

pairs of requests; we call the new algorithm *Seeded Genitor*. We compare the results of random initializations (Table 1) to those of running *GreedyStartLS* and *Seeded Genitor* (shown in Table 3). The initialization from a greedy permutation induces a major improvement in the local search performance. Initializing the *Genitor* population also results in major improvements on the results obtained for the new days of data (in fact, except for R1 all the results are equal to the best known results). Also, while SWO is still better than *GreedyStartLS*, *Seeded Genitor* finds equal or better solutions that does SWO.

Multiple Moves in the Search Space

We hypothesize that multiple moves are needed to effectively traverse the search space to good solutions. To test this, we implement a version of the SWO where only the request that contributes the most to the sum of the overlaps is moved forward. We use a distance proportional to the contribution of the chosen request to the sum of overlaps. We call this new algorithm SWO1Move. The results obtained by running SWO1Move for 30 runs, with 8000 evaluations per run are presented in Table 4.

The performance of SWO worsens significantly when only one task is moved forward. In fact, *GreedyStartLS* outperforms SWO1move, suggesting just focusing on the major contributors to the evaluation error is not enough. These results support the conjecture that the performance of SWO is due to the simultaneous moves of requests.

Progress toward the Solution

To further compare the genetic algorithm and SWO, we track the best value obtained so far when running the two algorithms. For each problem, we collect the best value found by SWO and *Seeded Genitor* in increments of 100 evaluations for up to 8000 evaluations. We average these values over 30 runs of SWO and *Seeded Genitor*, respectively. A typical example is presented in Figure 1. In the beginning, SWO progresses very fast to a good solution, but further improvements over time are small. On the other hand, *Seeded Genitor* steadily progresses in smaller steps toward the best solution, and while it takes longer to reach values as good as the ones produced by SWO, it outperforms SWO given enough evaluations.

Conclusion

For AFSCN oversubscribed scheduling, directed leaping appears to be an effective strategy for traversing its large search space. Both the genetic algorithm and SWO make long leaps in the search space and quickly arrive at good solutions. These leaps may change as much as 50% of the schedule at each iteration. By contrast, local search, by only moving one task at a time, results in a slow creep through the search space. One explanation for the relatively poor performance of algorithms that creep is that large plateaus are present in the search space.

We investigate the role of initializing the search closer to the best solution (as in SWO). We find that initialization helps. However, the multiple moves are more important: the performance of SWO is significantly worse when only one of the conflicting tasks is moved.

Finally, we show that *Genitor* and SWO follow different paths through the search space. SWO finds good solutions fast. Given more evaluations, *Genitor* finds better solutions.

The success in transferring the flexibility measure from Air Mobility Command scheduling suggests the domains may have a great deal in common. Future work will investigate whether our results generalize to other oversubscribed scheduling problems such as AMC.

Acknowledgments

This research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-03-1-0233. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Adele Howe was also supported by the National Science Foundation under Grant No. IIS-0138690. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We wish to thank the AAAI reviewers for their insightful comments and an anonymous reviewer of a previous paper for, among other excellent advice, suggesting we take a second look at SWO.

References

Barbulescu, L.; Howe, A.; Watson, J.; and Whitley, L. 2002. Satellite range scheduling: A comparison of genetic, heuristic and local search. In *Proceedings of The Seventh International Conference on Parallel Problem Solving from Nature(PPSNVII)*.

Barbulescu, L.; Howe, A.; Whitley, L.; and Roberts, M. 2004a. Trading places: How to schedule more in a multi-resource oversubscribed scheduling problem. In *Proceedings of the International Conference on Planning and Scheduling*. to appear.

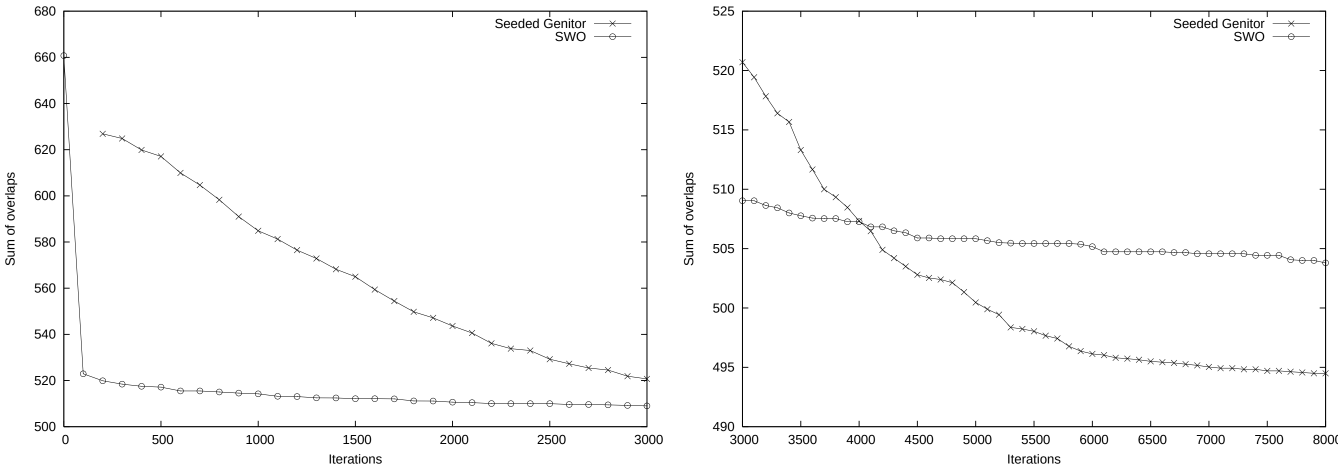

Figure 1: Evolutions of the average best value obtained by SWO and *Genitor* during 8000 evaluations, over 30 runs. The left figure depicts the improvement in the average best value over the first 3000 evaluations. The last 5000 evaluations are depicted in the right figure; note that the scale is different on the y-axis.

Barbulescu, L.; Watson, J.; Whitley, D.; and Howe, A. 2004b. Scheduling space-ground communications for the Air Force satellite control network. *Journal of Scheduling* 7(1):7–34.

Beck, J. C.; Davenport, A. J.; Sitarski, E. M.; and Fox, M. S. 1997. Texture-based Heuristic for Scheduling Revisited. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 241–248. Providence, RI: AAAI Press / MIT Press.

Bresina, J. 1996. Heuristic-Biased Stochastic Sampling. In *Proceedings of the Thirteenth National Conference on Artificial I ntelligence*, 271–278.

Deale, M.; Yvanovich, M.; Schnitzuius, D.; Kautz, D.; Carpenter, M.; Zweben, M.; Davis, G.; and Daun, B. 1994. The Space Shuttle ground processing scheduling system. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. Morgan Kaufmann. 423–449.

Frank, J.; Jonsson, A.; Morris, R.; and Smith, D. 2001. Planning and scheduling for fleets of earth observing satellites. In *Proceedings of the Sixth International Symposium on Artificial Intelligence, Robotics, Automation and Space*.

Gent, I., and Walsh, T. 1993. Toward and Understanding of Hill-climbing Procedures for SAT. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*. AAAI Press.

Gent, I., and Walsh, T. 1995. Unsatisfied variables in local search. In *Hybrid Problems, Hybrid Solutions*, 73–85. IOS Press Amsterdam.

Globus, A.; Crawford, J.; Lohn, J.; and Pryor, A. 2003. Scheduling earth observing satellites with evolutionary agorithms. In *International Conference on Space Mission Challenges for Information Technology*.

Gooley, T. 1993. Automating the Satellite Range Scheduling Process. In *Masters Thesis*. Air Force Institute of Technology.

Joslin, D. E., and Clements, D. P. 1999. “Squeaky Wheel” Optimization. In *Journal of Artificial Intelligence Research*, volume 10, 353–373.

Kramer, L., and Smith, S. 2003. Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems. In *Proceedings of 18th International Joint Conference on Artificial Intelligence*.

Lemaître, M.; Verfaillie, G.; and Jouhaud, F. 2000. How to manage the new generation of Agile Earth Observation Satellites. In *6th International SpaceOps Symposium (Space Operations)*.

Parish, D. 1994. A Genetic Algorithm Approach to Automating Satellite Range Scheduling. In *Masters Thesis*. Air Force Institute of Technology.

Smith, S., and Cheng, C. 1993. Slack-based Heuristics for Constraint Satisfaction Problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, 139–144. Washington, DC: AAAI Press.

Starkweather, T.; McDaniel, S.; Mathias, K.; Whitley, D.; and Whitley, C. 1991. A Comparison of Genetic Sequencing Operators. In Booker, L., and Belew, R., eds., *Proc. of the 4th Int'l. Conf. on GAs*, 69–76. Morgan Kaufmann.

Syswerda, G. 1991. Schedule Optimization Using Genetic Algorithms. In Davis, L., ed., *Handbook of Genetic Algorithms*. NY: Van Nostrand Reinhold. chapter 21.

Vaessens, R. J. M.; Aarts, E. H. L.; and Lenstra, J. K. 1996). Job shop scheduling by local search. *INFORMS J. Comput.* 8(3):302–317.

Whitley, L. D. 1989. The GENITOR Algorithm and Selective Pressure: Why Rank Based Allocation of Reproductive Trials is Best. In Schaffer, J. D., ed., *Proc. of the 3rd Int'l. Conf. on GAs*, 116–121. Morgan Kaufmann.

Wolfe, W. J., and Sorensen, S. E. 2000. Three Scheduling Algorithms Applied to the Earth Observing Systems Domain. In *Management Science*, volume 46(1), 148–168.