The Augmented Regret Heuristic for Staff Scheduling

Philip Kilby

CSIRO Mathematical and Information Sciences,
GPO Box 664, Canberra ACT 2601, Australia
August 2001

Abstract

The "regret" heuristic is a fairly well-known general-purpose construction heuristic. In problems where an initial solution must be constructed incrementally, there arises the problem: which variable do I set next, and to which value?

The regret heuristic proposes that at each stage the variable with maximum regret is selected. "Regret" is defined as the difference between the best and next-best outcomes from assigning a value. In this way, a variable that has only one "good" value remaining is assigned that value, ahead of a variable with many "good" values.

However a major problem with the heuristic is that it is myopic: it does not take into account any assignment beyond the next-best. This can lead to poor performance on some problems.

The *augmented regret* heuristic is an attempt to make the regret heuristic more robust. I will describe the new heuristic, and report its application to a staff scheduling problem, where it is more effective than the traditional regret heuristic. I will also discuss other problems where the new heuristic may also be effective.

1 Introduction

When a combinatorial optimization problem is too hard to solve optimally, or where a solution is required quickly, a common heuristic approach is to construct an initial solution then improve this solution. There has been much work on general methods for improvement, and meta-heuristics such as Tabu Search ([8, 9]), Simulated Annealing [5] and the like have been proposed.

This paper looks at the construction phase of such a method. *Regret* is a general-purpose heuristic used in the construction phase of some methods. For example [3] uses regret in the construction of solutions to the Travelling Salesman Problem.

The traditional regret heuristic looks at the difference in cost between the most preferred and next most preferred choices for an action - a large difference (large regret) means more urgency for that action.

However, by looking at only the first two choices Regret suffers problems when neither of these choices can be made and a more expensive option must be taken. The idea of the new heuristic – *Augmented Regret* – is to try to give regret a more global perspective. It

does this by allowing the costs of even less-preferred choices to influence the selection of the current most-preferred choice.

It is hoped that the Augmented Regret heuristic will prove to be a useful general-purpose construction heuristic. However at this stage we have only examined its use in the context of a Staff scheduling problem. The main purpose of this paper is to show that within this context Augmented Regret is better than the traditional Regret heuristic.

"Regret" has a fairly narrow definition in this paper: it is using the difference between the best and next-best choice as a guide to choosing an action. Concepts called "regret" occur in other contexts, for example *Minimax regret* is a common heuristic used in the context of Location problems [1, 4, 6]. These uses of the term are outside the context considered here.

I start with a more formal description of the usual Regret heuristic, and describe some of its problems. In Section 3 I describe the new heuristic. I then describe the Staff Scheduling problem we are solving, and report (in Section 5) on computational tests using both traditional Regret and the new Augmented Regret heuristics.

2 Traditional Regret

I will describe the regret heuristic in terms of staff allocation. Say we wish to allocate jobs to staff: some staff are better qualified to do a job than others, but each staff member may only do a certain number of jobs.

Look at the problem from the point of view of a particular job - say job i. We can assess the "cost" c_{ij} of assigning job i to each employee j. The lower the cost, the better qualified is the employee to do that job.

We can sort the c_{ij}'s for a job to get an ordering of employees - say $c_{ij_1}, c_{ij_2}, ...$, so that best employee for the job is employee j_1, with minimum cost c_{ij_1}. The regret heuristic looks at the difference between the cost of assigning the best employee and the next best employee - i.e.

$$\text{regret}_i \;=\; c_{ij_2} - c_{ij_1} \tag{1}$$

A large value of regret_i means the cost of not getting job i's most preferred employee is high. Hence,the job with the largest regret should be given their preferred employee.

The regret heuristic in this context would proceed by selecting the job with the maximum regret and assigning it to its most preferred employee. It would then re-calculate the costs (and feasibility) of assigning each job to each employee, and iterate.

Pseudo-code for the traditional regret heuristic can be found in Figure 1. It is described in terms of objects $i \in I$ and actions $j \in J$[1] The cost of performing action j on object i is c_{ij}.

A small numerical example will show the operation of the algorithm and a significant problem with the traditional algorithm. Say we have 4 tasks and 4 employees, each

[1]Assume for ease of exposition that the actions that can be performed on an object are the same for all objects. The method can be easily modified for the case where actions are object-dependent.

```
set I' = I
while I' not empty
    foreach i ∈ I'
        set numLegal_i = 0
        foreach j ∈ J
            if action j is legal for object i
                set c_ij = cost of performing action j on object i
                set numLegal_i = numLegal_i + 1

        if (numLegal_i == 0)
            error: feasible solution not found
        else if (numLegal_i == 1)
            regret_i = ∞
            action_i = j_1
        else
            sort c_i. to give c_ij1, c_ij2, ...
            regret_i = c_ij2 − c_ij1 (A)
            action_i = j_1

    find i* | regret_i* = max_i regret_i
    implement action action_i* on object i*
    I' = I' \ {i*}
```

Figure 1: The traditional regret heuristic

employee may perform only one task. The costs of assigning tasks to employees is:

		Employees			
		A	B	C	D
Tasks	1	12	13	2	1
	2	6	5	1	4
	3	5	9	1	3
	4	12	11	1	1

		Sorted by cost			
Tasks	1	1/D	2/C	12/A	13/B
	2	1/C	4/D	5/B	6/A
	3	1/C	3/D	5/A	7/B
	4	1/C	1/D	11/B	12/A

		Regret
Tasks	1	1
	2	3
	3	2
	4	0

So task 2 would be assigned to its first choice, employee C (cost 1). We can no delete column "C" (since C has been assigned a job, and row 2. This gives the remaining regrets:

		Regret
Tasks	1	9
	3	2
	4	10

So task 4 is assigned employee D (cost 1). With columns "C" and "D" deleted, the remaining regrets are

		Regret
Tasks	1	1
	3	2

And task 3 is assigned employee A (cost 5), leaving task 1 with employee B (cost 13). Total cost 20.

The optimal solution is 1–D, 2–B, 3–A, 4–C, total cost 12.

This demonstrates the shortcoming of traditional regret: by taking a short-term view – only considering the first two costs – it cannot "see" the approaching disaster of having to make the assignment 1–B. This motivates the new heuristic.

3 Augmented Regret

In the Augmented Regret algorithm the "regrets" of future decisions are also incorporated, but discounted the further away the decision. A simple linear discount is used. As in Section 2 let $c_{ij_1}, c_{ij_2}, ...$ be the sorted list of costs and assuming there are K_i legal actions for object i, the definition of simple augmented regret is

$$\text{simpleAugRegret}_i \;=\; \sum_{k=1}^{K-1} \frac{(c_{ij_{k+1}} - c_{ij_k})}{k} \tag{2}$$

The first term in this sum is the same as traditional regret. Subsequent terms factor in the potential cost of other actions.

Augmented regret is, if you like, a more pessimistic heuristic: rather than being forced to take the next best choice, it imagines scenarios where even less desired choices are made.

Simple augmented regret can be further refined by looking at the number of legal choices. In some situations different objects may have different numbers of legal actions. An object with only two remaining legal actions is surely more important than an object with many legal actions – even if those two actions have the same cost, and hence would have regret 0 under bother traditional and simple augmented regret.

To handle this situation augmented regret adds another term. The term relies on *maxCost* $= \max_{i,j} c_{ij}$; and *augMult*, a tuning parameter.

$$\text{augRegret}_i \;=\; \sum_{k=1}^{K-1} \frac{(c_{ij_{k+1}} - c_{ij_k})}{k} + (|J| - |K|) \times \mathit{maxCost} \times \mathit{augMult} \tag{3}$$

This is the simple augmented regret plus an extra term. The extra term adds in (*maxCost*× *augMult*) for each illegal action - hence the fewer legal actions the greater the regret score. The parameter *augMult* can be used to tune the algorithm. A larger *augMult* gives more importance to the number of legal actions, and hence should be used when finding a feasible solution is is difficult.

This approach of looking at the number of feasible actions is analogous to the "first-fail" heuristic from Constraint Programming (see [7] for example).

The simple augmented regret and augmented regret heuristics operate in the same way as the the traditional regret described in Figure 1, except that the expression labeled "(A)" is replaced by either equation 2 or equation 3.

4 A problem from staff scheduling

The problem we will consider in computational testing is an extension of the one used in examples above. It is drawn from "real life" and is the problem that motivated the research.

Employees must be assigned to tasks. Each task must be assigned. Each task has attributes of start day and time, and a fixed duration.There are rules – in the form of both hard and soft constraints – governing the allocation. Two forms of cost are associated with an allocation: a fixed cost c_{ij} of allocating task i to employee j; plus a variable "penalty" cost, derived from the soft constraints, which depends on the other jobs assigned to an employee.

Different employees have different aptitudes for each job, based on the employee's score in three attributes. Each employee has a random value for the three attribute scores. Each job has a desired level of each of these attributes. The employee's aptitude is a measure of how many attributes s/he fulfills, and how close s/he is to the attributes s/he falls short on. That is, for each attribute, the employee scores 1.0 if *level* $\geq$ *required level*, and $\frac{\textit{required-level}-\textit{level}}{\textit{required-level}}$ for each attribute where *level* $<$ *required level*. The scores from the three attributes are summed an then normalized to a value between 0 and 1. These costs are calculated *a priori*.

The hard constraints modelled here are:

- Each employee can perform at most one task per day
- There is a limit on the total work time assigned to an employee
- There is a minimum gap between tasks
- There is a minimum number of "days off" – days where no task is assigned – over the duration of the problem
- There is limit on the number of consecutive work days

In addition there are "soft constraints" in the form of penalties applied to improve the solution. These are

- A minimum work time
- A cost which minimizes the number of staff used
- A cost which encourages "weekends" for staff - i.e. two consecutive rest days

The costs associated with soft constraints depend on the other tasks assigned to an employee, and so cannot be calculated *a priori*.

5 Tests

Two sets of random test problems have been generated for this class of problem. Since no exact solution is available, the new construction heuristics are compared to the the solution generated by running several improvement operators.

The constraints are common to both problem set: the work day is divided into three 8-hour shifts. At most 80 hours work can be assigned over the 14 day horizon. The minimum time between shifts is 8 hours. A minimum of 2 days off is required over the 14 days. A maximum of 6 consecutive work days is permitted. The target for minimum work time is set at 24 hours.

The characteristics of the problem first problem set are as follows:

- Horizon of 14 days
- 200 jobs
- 30 staff

One hundred test problems of this type were generated. Because no exact solution procedure is available, a "Best Known" solution was obtained by running several improvement heuristics from several starting solutions.

The second set of problems is designed to be harder to solve. These are the same 100 test problems as for problem set 1, but only 22 staff are available. This means that some problems are infeasible – of the 100 test problems no solution could be found for 14 problems. The tables below report the statistics for the remaining 86 problems from problem set 2.

6 Results

The algorithms tested were

Regret The traditional regret heuristic.
AugRegret The simple augmented regret heuristic
AugRegret+1.0 The augmented regret heuristic with a value of 1.0 for *augMult*
AugRegret+2.0 The augmented regret heuristic with a value of 2.0 for *augMult*
AugRegret+3.0 The augmented regret heuristic with a value of 3.0 for *augMult*
AugRegret+4.0 The augmented regret heuristic with a value of 4.0 for *augMult*
BestKnown The best result found from five runs of an improvement heuristic

Table 1 presents the results for problems in problem set 1 (described in Section 5). The columns of this table give:

Percent Problems Solved For how many of the 100 problems was this method able to produce a feasible solution?

Num Times Min For how many of the 100 problems did this algorithm produce the best result amongst the *construction* algorithms. This column compares construction heuristics only; the "BestKnown" value from improvement heuristics is not considered.

Algorithm	Num Problems Solved	Num Times Min	Percent Above Min	Std Dev
Regret	100	1	20	14
AugRegret	100	30	15	11
AugRegret+1.0	100	27	15	12
AugRegret+2.0	100	26	15	10
AugRegret+3.0	100	23	15	11
AugRegret+4.0	100	22	15	10
BestKnown	100	–	0	0

Table 1: Results for Problem Set 1 (100 problems)

Algorithm	Num Problems Solved	Num Times Min	Percent Above Min	Std Dev
Regret	26	3	20	8
AugRegret	23	13	13	7
AugRegret+1.0	23	13	13	7
AugRegret+2.0	27	17	14	8
AugRegret+3.0	33	11	16	9
AugRegret+4.0	23	8	16	11
BestKnown	86	–	0	0

Table 2: Results for Problem Set 2 (86 problems)

Percent above Min The average percent above the "Best Known" value

Std Dev The standard deviation of the "Percent above Min" statistic, to give an idea of the variability of the method.

Table 2 presents the results for the more difficult problems – in terms of finding a feasible solution – of problem set 2. The columns of this table are as for Table 1 except that only 86 problems allow a feasible solution.

Table 1 shows that for these test problems where finding a feasible solution is not so difficult, both the Simple Augmented Regret heuristic and the Augmented regret heuristic with all 4 values of *augMult* find solutions which are on average 5% better than those of the traditional regret heuristic. Traditional regret found the best solution (amongst the construction heuristics) only once in the 100 problems; one of the Augmented heuristics found the best solution in the remaining cases.

It is interesting to look at the relative performance of the Augmented heuristics. The Simple Augmented regret heuristic was the best construction heuristic most often, all values of *augMult* produced solutions of similar quality and variability.

Table 2 shows the results where finding a feasible solution is more difficult. As expected, a higher value of *augMult* results in better ability to find a solution - from 23 (out of 86) problems solved for Simple Augmented Regret, through to 33 problems solved for Augmented Regret with *augMult* set to 3.0.

Setting *augMult* to 4.0 results in a much poorer performance – both in terms of number of problems solved and solution quality. Obviously there is a point at which the augmenting term swamps the regret value, and the heuristic starts to breaks down. This implies that

the value of *augMult* must be chosen carefully for these more difficult problems.

Traditional regret solves 26 problems – well short of the best-performing Augmented algorithm. In terms of solution quality, traditional regret is again approximately 5% behind Augmented Regret (with *augMult* 3.0) when compared over problems where both algorithms found a solution.

7 Conclusions

I have described a new construction heuristic – the Augmented Regret heuristic – which attempts to improve upon the traditional Regret heuristic. The Augmented Regret heuristic uses a regret score which is the result of combining the "regrets" from all possible values of a variable. I claim that this makes the heuristic more robust and more reliable than the traditional algorithm.

I have demonstrated the effectiveness of the new heuristic using examples from a staff scheduling problem. Testing on two sets of problems has shown that the new algorithm is able to outperform the traditional heuristic both in terms of the number of difficult problems it can solve, and the solution quality of both hard and easier problems.

The next step for this research is to test the algorithm in other problem areas, to validate its usefulness as a general-purpose heuristic.

Because the heuristic is a general-purpose heuristic, I do not expect that it will perform well compared to heuristics purpose-built for a particular problem area, and which incorporate detailed domain knowledge. Also, it seems to perform best in situations where feasible solutions are difficult to find, or where differentiation based on the number of feasible values for each variable is important. For example, I would not expect the Augmented Regret heuristic to be much better than the traditional regret on solving Travelling Salesman Problems ([10]): any tour is feasible, and feasibility of placement it not a strong characteristic. In contrast, I think the heuristic should be effective in Time-Window constrained Vehicle Routing Problems ([2]) as these characteristics are present. This is the next problem area I intend to investigate.

References

[1] I. Averbakh. Minimax regret p-centre location on a network with demand uncertainty. *Location Science*, 5(4):247–254, 1997.

[2] Dimitris J. Bertsimas and David Simchi-Levi. A new generation of vehicle routing research: Robust algorithms, addressing uncertainty. *Operations Research*, 44(2):286–303, 1996.

[3] Yves Caseau and Francois Laburthe. Solving small TSPs with constraints. In *International Conference on Logic Programming*, pages 316–330, 1997.

[4] Mark S. Daskin, Susan M Hesse, and Charles S. Revelle. α-reliable P-minimax regret: A new model for strategic facility location modeling. *Location Science*, 5(4):227–246, 1997.

[5] Kathryn A Dowsland. Simulated annealing. In Colin R Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, chapter 2. Blackwell Scientific Press, Oxford, UK, 1993.

[6] Zvi Drezner and Jeffery Guyse. Application of decision analysis techniques to the weber facility location problem. *European Journal of Operational Research*, 116:69–79, 1999.

[7] I.P. Gent, E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Proceedings of the 2nd International Conference on the Principles and Practice of Constraint Programming CP '96*, pages 179–193, 1996.

[8] F. Glover. Tabu search, part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[9] F. Glover. Tabu search, part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[10] Gilbert Laporte. The traveling salesman problem: an overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:231–247, 1992.