Bakery Quarrels - Backtrack Versus WalkSat

Introduction to Artificial Intelligence – Final Project

Ido-Moshe Silverwater Noy Sternlicht Jonathan Weiss

***Abstract*— Modelling a problem as a constraint satisfaction problem isn't always possible: The problem may contain different type of constraints and we would like to solve as many as we can instead of stating that the problem is not solvable.**

I. INTRODUCTION

In this paper we wish to show a problem that isn't easily modelled as a constraint satisfaction problem, and our method of solving it.
In order to solve the problem we tried modelling it into a 'soft-hard' constraint satisfaction problem and used a modified backtrack solver. The second method was to translate the CSP representation into a logical CNF formula to be satisfied by a walkSat solver.

II. THE PROBLEM

We modelled a worker — employer situation: Creating a schedule for the workers composed of days and shifts. each day has *m* shifts, every shift should have at least 1 workers assigned to it, and at most n. The workers may have preferred shifts in the upcoming week, and shifts they cannot take because of other obligations. Lastly some workers can demand an amount of shifts in the coming work week.
This problem isn't necessarily satisfiable furthermore we wish to satisfy as many workers as we can. As a result it is not a problem that can be modelled by a CSP, and thus cannot be solved using a solver such as the backtrack algorithm.
We represented the problem using a program that reads the wishes of the employees (given a specific format) and generates a sequence of constraints of two different and distinct categories:

A. Hard Constraints

The 'main' constraints, These type of constraints aren't to be ignored and must be satisfied. Examples of these constraints are: There must be at least one worker in a shift. There cannot be a worker that will be assigned into a shift that he specified he cannot work at.
There cannot be more than *n* workers.
These constraints are represented by 0 softness level in our program.

B. Soft Constraints

The soft constraints can have many levels in-between them, they are represented by an ascending softness level where 1 is the least soft. These constraints are considered "nice to have" where we aren't forced to satisfy them. These constraints can represent the preferred work days of each worker.

Assumptions:

We assumed that every worker can do the same job inside the bakery, as a result we didn't distinct between the workers. This can be modelled too as an advanced constraints class deriving from our implemented class.

III. DESCRIBING THE PROBLEM AS CSP

We created the class CSP which holds all the variables, and a constraints class. Each variable is responsible for holding it's own domain, and the set of neighbours it connects to. Each constraint object remembers the variables $v_1, v_2, \ldots, v_n$ that it is tied to, and a list of possible assignments for all the variables combined - that is the set:
$\{(d_1, d_2, \ldots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n\}$ where $D_1, \ldots D_n$ are the possible domains of variable $v_1, v_2, \ldots, v_n$.
Possible domain meaning the domain of the variable that is allowed. After forward checking the possible domain of a variable may change for example. The Constraints class is responsible of holding and managing all the constraints. Finally the CSP is given to a solver which uses it's functionality to return some kind of solution.
Because we decided to use as a method a logical solver we created the domain of every variable to be $\{\mathbb{T}, \mathbb{F}\}$ and we defined the variables according to the workers names to be:
$< name >< day >< shiftNum >$.

IV. SOLVERS:

We created 3 solvers to face the problem:

A. Backtrack

We implemented the backtrack algorithm, and a few heuristics to be used by it: Degree, Least constraining value, Minimum conflict, Minimum Remaining Value.
The backtrack had a hard problem solving even for a 2 employee problem. As a result we implemented forward checking which eliminates domain values for the variables and limiting the options overall. Using the forward check we managed to eliminate the enormous runtime of most of the problems we faced.
We modified the backtrack algorithm such that it first tries to satisfy only the hard constraints, if it isn't possible than the backtrack returns no assignment and declares failure. If the backtrack managed to satisfy the hard constraints, we start adding the soft constraints one by one and after each constraint added - we try to satisfy. This is a greedy approach in order to satisfy as many soft constraints as we can. We choose the next soft constraint to add to the constraints that the backtrack looks at using a soft heuristic chooser:

We created 4 different heuristic for the mission of choosing the next soft constraint.
The common thing between the first 3 heuristics is that each method sorts between the levels of 'softness' first and than sorts for each level differently. The heuristics:
Degree heuristic - checks how much neighbours a soft constraint have to the hard constraint and choose the one with the least. Name heuristic - looks at the amount of variables the soft constraint is tied to, and chooses the one that has the least different variables in it. Assignment heuristic - we choose the constraint that has the most amount of possible assignments.
The last heuristic is just a random shuffle over the soft constraints.

Take heed that the backtrack's runtime is heavily influenced by the sorting of the variables - hence the heuristics. A good sorting may result in a lighting fast operation. Our heuristics used sorting algorithms and data that were contained in hash maps, as a result every run may have different runtime.

B. *Logical WalkSat*

We translate the constraints representation into logical representation. To do so we generated trees that represent the relationships between the assignments and constraints as follows: We added a logical and between each constraint. and every constraint clause is created by a logical or between every assignment. every assignment is a logical and between the literals which are actually the possible values for a variable. these are our literals. After creating a tree representing all the constraints in a logical formula, we implemented a formula to CNF function that should've create a CNF formula out of the tree. Using this CNF formula we invoke a walkSat solver over the formula. the returned assignment should be the output of the walkSat.

Notice that the walkSat is an approach of solving the problem, but as the nature of walkSat - it isn't obligated to give a complete solution. If it couldn't find a solution it will just result in a solution that might and might not satisfy the constraints. WalkSat is allowed to return an assignment that does not fully satisfy all hard constraints.

Unfortunately we did not manage to get results from this method because the computational and space needed was a task to hard for our computers. The reason is that converting the clauses to logical representation created an exponential number of clauses which resulted in exponential amount of clauses even for mere 7 employees it was too much. This was a problem in the amount of memory we needed to represent the clauses, As a result we do not have evaluation for the logical walkSat.

C. *CSP WalkSat:*

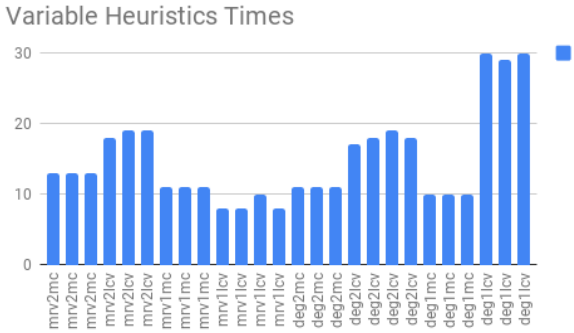
Because the Logical walkSat had a problem running we tried another approach of doing the walkSat algorithm to solve our problem: We defined the constraints as our clauses (again), but instead of creating clauses and generating a CNF formula we simply look at our constraints as a cluster (meaning each constraint hold a couple of clauses) of clauses that we wish to satisfy, and the variables tied to each constraint as the literals of a clauses.

V. RESULTS

Brief explanations and results comparing using charts to create analysis.

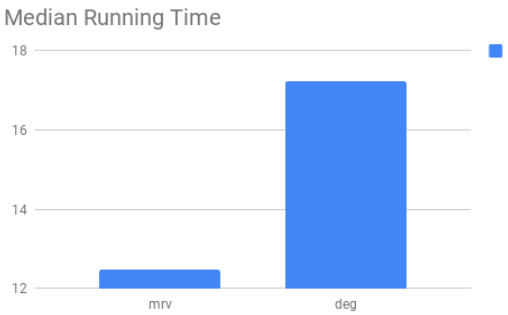
A. *Backtrack Parameters Analysis:*

We compared between the inner parameters of each solver. That is we compared between heuristics, and compared between the walkSat's probability and maximum flips allowed. We took the best run out of several runs of the backtrack.



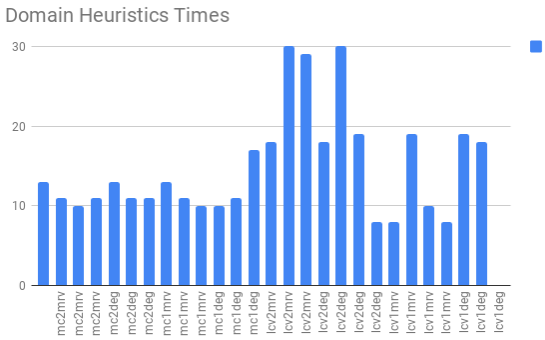
This chart shows the running time of backtrack using different heuristics on two different tests. The x axis represents the variable heuristic, test and domain heuristic as follows <variable heuristic><test number>< domain heuristic>. mrv- minimum remaining value, deg - degree, lcv - least constraining value, mc - minimum conflict. Test 1 and test 2 are two different random produced tests.

- Results may repeat because of data from other tests.



The results of the running time in seconds for several random generated examples. As we can see, the mrv as a whole is better suited to our problem.

From these results we understand that minimum remaining conflict is best suited as a variable heuristic to our problem.

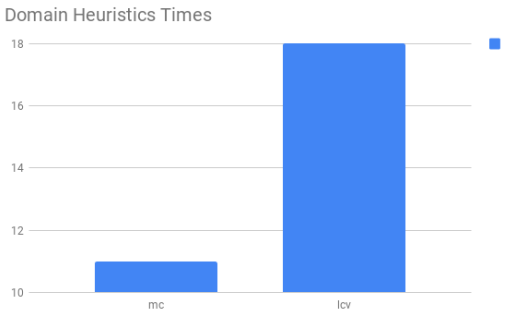


Running time in seconds using different heuristics to choose the domain.

Following is a comparison between domain heuristics:

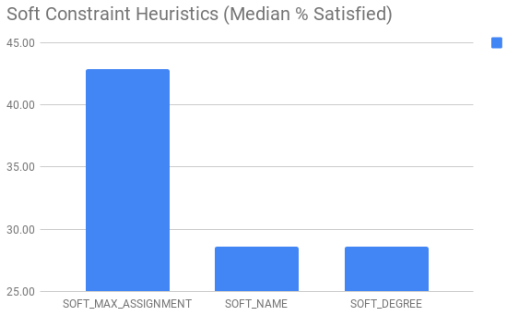
And the median running time:
It is understood from these comparisons that in most runs the minimum conflict is wanted as a domain heuristic.

Afterwards we looked upon the soft constraint heuristics,



Median running time in seconds.

that is the heuristic to order soft constraints for the backtrack algorithm.

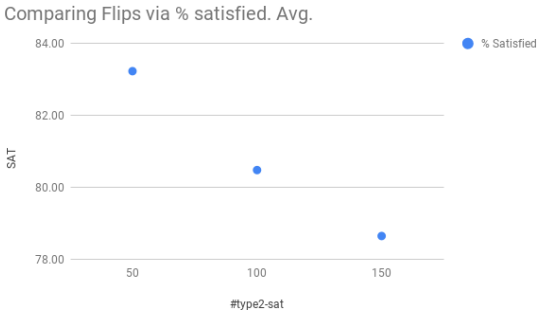


SOFT_MAX_ASSIGNMENT means the heuristic that chooses the next soft constraint according to the maximum of possible assignments it has. SOFT_NAME is counting the amount of variables this heuristic is tied to. SOFT_DEGREE chooses according the constraint with the minimum hard constraints tied to.

We can see from the chart above that max assignments is the preferred heuristic for choosing the next heuristic to add, the reason is that adding a heuristic with a large amount of possible assignments limits the least the current satisfaction problem.

B. *WalkSat Parameters Analysis:*

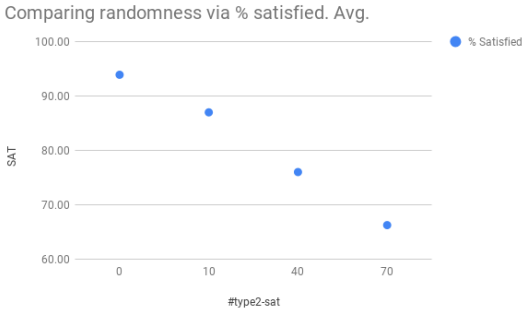
We start by observing how many flips after the initial random assignment achieves good results:



The chart above is the results of the average of many tests with different parameters. (Using 25 different tests for each probability.)
The X axis represents the amount of flips that were used.
The Y axis represent the percentage of satisfied constraints overall.

It is noticeable from this graph that 50 flips is a desired parameter over 100 and above, but on further inspection we found out the less than 50 flips is less optimal as well. As a result for this problem we should use 50 flips.

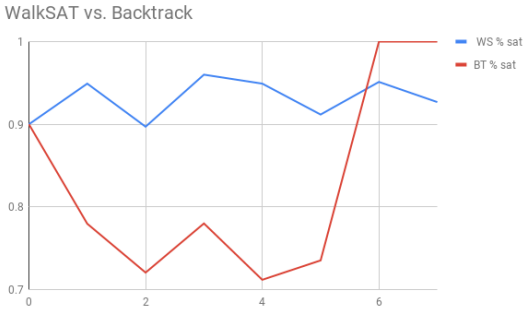
To our surprise we found out as the graph that follows shows that choosing to be greedy alone is better than being random in our simulated problems. We deduce from these results that our problem is best solved greedily with a walkSat rather than just trying to unbind a bad situation using randomness.



The X axis represents the probability of being greedy or random in the walkSat.
The Y axis represent the percentage of satisfied constraints overall.

We assumed there is a some solution in the system that random initialisation puts us close enough to it, and being greedy we can just get closer.

C. *Backtrack Vs WalkSat:*



The red line and blue line represent the backtrack and walkSat results respectively.
the Y axis is the percentage of satisfied constraints including soft and hard constraints. take to mind that not all of the constraints can be satisfied together. The X axis represent a different test each.

We tested with different test files both algorithms, and made a chart to describe the results.
it can be seen that when there isn't necessarily a satisfying assignment, walkSat is superior in satisfying soft constraints, although it isn't necessarily satisfying all hard constraints.
the spike for the red line is determined by a problem that is fully satisfiable and as a result the backtrack algorithm can solve it complete, as oppose to the walkSat.

The full results are in another file, for the convenience of the reader[1]

The tests will be added to the project files and can be inspected.

VI. CONCLUSIONS

Looking at the results we conclude that for this specific problem, most of the time the walkSat is more forgiving to the hard constraints, as a result they aren't always satisfied and that is a bit of a problem, on the other hand the

[1] https://docs.google.com/spreadsheets/d/e/2PACX-1vRYuay_b_UhERDIoW5didfuVxBeZW7cTEG8f2wYXCO_LVMPLHe4XijmeTUV_-ROuA9FbsQJioTS5Ns4/pubhtml

backtrack can’t always output an assignment and or some kind of solution at all. Hence it has it’s downfalls too, especially in complicated problems with a handful of wishes and complaints from the employees.
It is best to combine the both in order to achieve the best results, if the hard constraints are indeed as we modelled them, an obligation - the backtrack should be chosen. on the other hand if we wish to consider the wishes of our workers it is possible to use walkSat as a first guess and fill in the missing details personally afterwards. In any approach this two tools can be applied to automate even a bit the process of assigning the shifts, and satisfying constraints.

As a future work it is possible to combine somehow both algorithms to achieve better results overall.