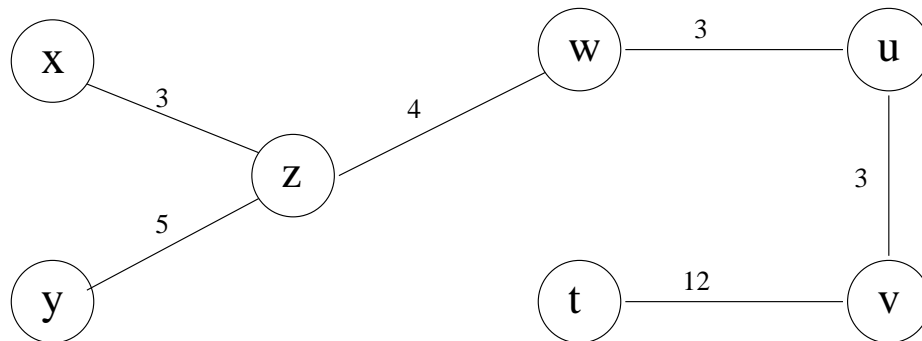


Solutions to problem set 3

Problem 1. Constraint satisfaction and Constraint Propagation

The CSP problem is represented by the following graph.



Let's summarize the assignments and constraints we have.

Equations: (init. assgn.)	$x = 2$ $y = 0$ $t = 0$	Constraints:	$x \equiv z \pmod{3}$ $y \equiv z \pmod{5}$ $z \equiv w \pmod{4}$ $w \equiv u \pmod{3}$ $u \equiv v \pmod{3}$ $v \equiv t \pmod{12}$
------------------------------	-------------------------------	--------------	---

Moreover, there are the constraints that $0 \leq x, y, z, w, u, v \leq 9$.

Equations are the formulas of the form $variable = value$. *Disequations* are the formulas of the form $var \neq value$.

Forward checking

$$\text{Equations: } \left\{ \begin{array}{l} x = 2 \\ y = 0 \\ v = 0 \\ t = 0 \end{array} \right\} \quad \text{Disequations: } \left\{ \begin{array}{l} \text{none} \end{array} \right\}$$

Forward checking infers: (a) disequations from equations and constraints, and (b) equations through the exhaustion of alternatives. Intuitively, the process of inferring a disequation corresponds to checking whether the new assignment of value to a variable (equation) restricts domains of variables connected to it in the constraint graph. These restrictions are represented as disequations (invalid assignments).

As an example, consider the assignment $x = 2$. According to the mod constraints, we can delete values 0,1,3,4,6,7,9 from the domain of z , to ensure the consistency. This is equivalent to inferring disequations $z \neq 0, z \neq 1, z \neq 3, \dots, z \neq 8$. So now, $\text{dom}(z) = \{2, 5, 9\}$.

$$\text{Equations: } \left\{ \begin{array}{l} x = 2 \\ y = 0 \\ v = 0 \\ t = 0 \end{array} \right\} \quad \text{Disequations: } \left\{ \begin{array}{l} z \neq 0, 1, 3, 4, 6, 7, 8 \end{array} \right\}$$

Now, assume equation $y = 0$. Forward checking along the second constraint forces us to delete 2 and 9 from $\text{dom}(z)$. Again, this means we have inferred the disequations $z \neq 2, z \neq 9$. Now, $\text{dom}(z) = \{5\}$ and we can infer that $z = 5$ – *by exhaustion of alternatives*.

$$\text{Equations: } \left\{ \begin{array}{l} x = 2 \\ y = 0 \\ v = 0 \\ t = 0 \\ z = 5 \end{array} \right\} \quad \text{Disequations: } \left\{ \begin{array}{l} z \neq 0, 1, 2, 3, 4, 6, 7, 8, 9 \end{array} \right\}$$

So far we have used equations $x = 2, y = 0$. We have two more equations we haven't evaluated yet, $v = 0$ and $z = 5$. By taking $v = 0$ we can delete 1,2,4,5,7,8 from the domain of u via forward checking, which gives:

$$\text{Equations: } \left\{ \begin{array}{l} x = 2 \\ y = 0 \\ v = 0 \\ t = 0 \\ z = 5 \end{array} \right\} \quad \text{Disequations: } \left\{ \begin{array}{l} z \neq 0, \dots, z \neq 9 \text{ (except 5)} \\ u \neq 1, 2, 4, 5, 7, 8 \end{array} \right\}$$

So, $\text{dom}(u) = \{0, 3, 6, 9\}$.

The equation $z = 5$ (inferred) triggers further constraint propagation. In particular we delete values 2, 3, 4, 6, 7, 8 of w as inconsistent with $z = 5$, or equivalently infer that $\text{dom}(w) = \{1, 5, 9\}$. The final state of the forward checking procedure is:

$$\begin{array}{l} \text{Equations:} \\ \left. \begin{array}{l} x = 2 \\ y = 0 \\ v = 0 \\ t = 0 \\ z = 5 \end{array} \right\| \end{array} \quad \begin{array}{l} \text{Disequations:} \\ \left. \begin{array}{l} z \neq 0, \dots z \neq 9 \text{ (except 5)} \\ u \neq 1, 2, 4, 5, 7, 8 \\ w \neq 0, 2, 3, 4, 6, 7, 8 \end{array} \right\| \end{array}$$

and now no further conclusions can be made.

Arc consistency

Observe that all inferences made in FC are inferences that would be also made by the arc consistency method. In addition, arc consistency procedure also checks the consistency of remaining values for pairs of variables along the arc that connects them. Let us consider variables w, u and arc $w \rightarrow u$.

The remaining values for w are $\text{dom}(w) = \{1, 5, 9\}$ and for u these are $\text{dom}(u) = \{0, 3, 6, 9\}$. Let us check if individual values of w are consistent with the remaining values for u . Let us first assume $w = 1$. Values of u that are consistent with $w = 1$ across mod 3 constraint are $\{1, 4, 7\}$. Unfortunately none of these values is in the current $\text{dom}(u) = \{0, 3, 6, 9\}$, hence the value $w = 1$ is inconsistent with u and we can infer $w \neq 1$. Now assume $w = 5$. This value is consistent with values of $\{2, 5, 8\}$ for u . Once again, none of these values is consistent with current $\text{dom}(u) = \{0, 3, 6, 9\}$, hence $w \neq 5$ can be inferred. Finally, let us try $w = 9$. The value is consistent with values 3, 6, 9. These values indeed overlap with the remaining values $\text{dom}(u) = \{0, 3, 6, 9\}$ hence the assignment is consistent. Taking the new disequations $w \neq 1$ and $w \neq 5$ together with the previously derived disequations, we can infer $w = 9$ through the exhaustion of alternatives. Now, all arcs are consistent (verify!) and the resulting partial assignment is:

$$\begin{array}{l} \text{Equations:} \\ \left. \begin{array}{l} x = 2 \\ y = 0 \\ v = 0 \\ t = 0 \\ z = 5 \\ w = 9 \end{array} \right\| \end{array} \quad \begin{array}{l} \text{Disequations:} \\ \left. \begin{array}{l} z \neq 0, \dots z \neq 9 \text{ (except 5)} \\ u \neq 1, 2, 4, 5, 7, 8 \\ w \neq 0, 2, 3, 4, 5, 6, 7, 8 \end{array} \right\| \end{array}$$

The only variable that remains unassigned is u . It is only partly constrained ($\text{dom}(u) = \{0, 3, 6, 9\}$) and no new equation or disequation can be inferred.

Problem 2. Traveling Salesman Problem

The Traveling Salesman problem (TSP hereafter) is a classical graph-theoretical problem. It involves a traveling salesman who has to visit each of the cities in a given network before returning to his starting point, at which time his trip is complete. The objective is to find the cheapest tour, that is the shortest route that passes through each city ones and returns back to start.

The variants of TSP arise in the design of telephone networks and integrating circuits, in planning construction lines, and in the programming of industrial robots, to mention but a few applications. In all of these, the ability to find inexpensive tours of the graphs in question can be quite crucial.

In this assignment, we explore **simulated annealing** solution to the TSP problem. The TSP we use consists of n cities placed on a two dimensional map. The (x, y) coordinates define locations of cities. The objective is to find a tour with the shortest overall length. The distance between the two cities is the Manhattan distance of their locations:

$$d_M(A, B) = |x(A) - x(B)| + |y(A) - y(B)|.$$

Part a. Simulated annealing algorithm

Simulated annealing explores the space of all tours by generating random rearrangements of the current tour. The new tour is accepted when its energy (distance) is smaller than the energy of the current tour. The tour with a higher energy is accepted randomly with probability $e^{\Delta E/T}$, where ΔE is the energy difference between current and new energy, and T is the temperature parameter that is changed during the search. The probabilistic choice is a solution to the problem of local optima.

Implement a simulated annealing function `sim_anneal(problem, init_temperature, no_of_steps)` for solving the TSP problem. All of the above simulated annealing code should be included in the file `main3a.c`. In addition, the code should run a simulated annealing code on the standard TSP problem with number of simulation steps = 100,000 and initial temperature = 1000. The standard TSP is given in file `std_tsp.txt` and consists of 60 cities. Use function `generate_standard_TSP` to read in the problem.

Solution. Programs for part a can be found on the course web page.

Part b. Experiments with the simulated annealing program

Experiment with your simulated annealing algorithm on the standard TSP problem (file *std_tsp.txt*) while varying parameters of the simulated annealing procedure: the initial temperature and the number of simulation steps. Choose values of the parameters such that your algorithm is able to find the solution with the path cost at least as small as 120. Submit the solution you have found, its distance (energy) and collected statistics in your report. See if you can beat our best solution of 72.892862. You do not have to submit programs you use in experiments in this part.

Solution. The answer to this problem varies quite a lot. Note that you have to make sure that you choose the tuple (T, k) in such a way as to allow the program to explore and try a variety of tours. If you start with small values of initial T , you essentially do hill climbing, which means that you are likely to get stuck in a local minima, and this independent of the number of steps k . So, with parameters like $(T = 0.1, k = 10000)$, we do not do much of the annealing and exploration of the configuration space, and even though we get a solution below 120, it is most likely the result of luck in selecting a good initial tour.

On the other hand, small number of steps k means that the overall number of configurations scanned is small and also that the annealing proceeds very fast. For example, with parameters set to $(T = 1000, k = 1000)$ the results were quite bad and solutions obtained were rarely better than 260. Even for $(T = 1000, k = 80000)$ we were consistently finding solutions only in the range of 145 – 185. This can be explained by a relatively fast cooling schedule with approximately 40000 accepted mutations overall.

You can usually get very good tours with final energy levels approx 80-110, (initial energy levels of approx 300-325) with parameters set to $(T = 200, k = 250000)$. For this case, the approximate number of mutations accepted is about 125000. The solution of 72.892862 given as the target to beat was obtained after few runs of the simulated annealing algorithm for parameters $(T = 200, k = 100000)$.

Part d. Experiments with the genetic algorithm

The disadvantage of the simulated annealing algorithm is that at any point in time it keeps only one current configuration and that next step configurations are obtained using “local” changes. Thus, it may take a number of steps till one gets to explore good configurations. The genetic algorithm attempts to alleviate the problems by keeping a limited number of “current” configurations and by combining (more radically) two good quality solutions hoping that the combination will lead to a larger improvement in the quality.

The default configuration is: Number of generations 300, Population size 300, Mutation probability 0.03, Survival rate 0.2. Please report best configuration(s) found by varying:

- (a) Number of Generations in the range between 50 and 500 in increments of 50 while keeping all other parameters in the default setting fixed.
- (b) Population size in the range between 50 and 500 in increments of 50 with the rest of the parameters set to defaults.
- (c) Mutation probability in the range between 0.00 and 0.25 in increments of 0.05 with the remaining parameters set to default values.
- (d) Survival rate in between 0 and 1 in increments of 0.2 with the remaining parameters set to default values.

Please report your findings either in tables (one for each experiment) or graphs. Analyze the results and draw the conclusions.

Solution.

The performance of the GA depends on all of the mentioned parameters. Along the dimensions of the parameter space, the following phenomena are observable.

- (a) Increasing the number of generations that the evolution is allowed to run has greatly improved the quality of solution. This is due to increased number of attempts to generate a good-fitness solution.

Number of Generations	Number of mutations	Best path evolved
50	733	161.988
100	1427	138.147
150	2140	121.639
200	2835	117.364
250	3610	106.102
300	4312	104.477
350	5152	102.294
400	5719	94.6874
450	6352	92.9236
500	7191	89.4075

- (b) Population size also influences solution quality. Again, larger population means more chances that the operators will generate a good solution.

For this problem, there appears to be little improvement beyond 100-150 individuals.

Population size	Number of mutations	Best path evolved
50	766	130.302
100	1457	106.548
150	2106	106.851
200	2954	105.56
250	3644	111.742
300	4385	116.394
350	5038	103.358
400	5837	110.515
450	6468	107.715

- (c) Mutation rate is crucial to GA's performance. A mutation rate that is too low leads to a very homogenous and stationary population. On the other hand, a mutation rate that is too high tends to break apart the co-evolved structures.

Observe the increase in number of mutations as the probability grows. Also observe, the evolution has not completely stopped, the crossover operation is still mixing the genes, although more slowly.

Rather large values of mutation probability seem to do our solutions good, the performance starts to deteriorate only around $p = 0.5$.

Mutation rate	Number of mutations	Best path evolved
0.00	0	159.808
0.01	1448	117.329
0.05	7144	95.5936
0.10	14288	93.3355
0.15	21518	85.0713
0.20	28755	93.5961
0.25	35976	92.5962
0.30	43073	84.3305
0.35	50515	79.7265
0.5	71906	96.21

- (d) Survival rate is important for preservation of well-evolved parents. If none of the parents were allowed to join reproduction in the next generation, it could happen that a good structure is forgotten. On the other side, if too many survive, the evolution is stalled. We see that low values of survival are appropriate for our problem.

Survival rate	Number of mutations	Best path evolved
0.0	5503	109.191
0.2	4304	111.554
0.4	3177	237.133
0.6	2137	266.36
0.8	1090	260.621
1.0	0	249.252

There is no “right” parent selection algorithm. A delicate balance between giving breeding advantage to the good individuals and preserving the variability in population should be kept. This is the one of the most difficult parts of a GA to design.