Report

Tianlang Tan
20028268
scytt1@nottingham.edu.cn

1. Description

This coursework is to solve Multi-dimensional knapsack problem by implementing memetic algorithm (MA) which is a variant of genetic algorithm. Memetic algorithm is combined with genetic algorithm (GA) and local search. In genetic algorithm, the main procedure is divided into encoding, initialization, selection, crossover, mutation and replacement. This basic idea of GA is to simulate natural selection that choose the better individual to survive. As for local search, it can find the local optimal in each generation. In this coursework, solution for Multi-dimensional knapsack problem is represented by binary representation. Initialization for all the solution are random. Tournament selection is selected as the selection operation. Uniform crossover is selected as the crossover operation. Mutation is operated randomly. Replacement is implemented to replace the best individuals to next generation. As for local search, this course uses variable neighborhood search as the local search method.

2. Pseudo code for memetic algorithm:

```
n: number of instances in the problem file
load the instances to Ins(i) (i = 0, 1, 2 ... n)
for each instance Ins(i)
memeticAlgorithm(Ins(i)){
    init_population(Ins(i), parent_pop); // initialize instance to a population
    while(iteration < itermax && time_spent < MAX_TIME){
        selection(mating_pool, parent_pop);  //tournament selection
        crossover(mating_pool);  //uniform crossover
        mutation(mating_pool);
        feasibility_repair(mating_pool); //keep droping the item with minimal price
 until the objective is within the constrain in all dimensions
        varaible_neighbourhood_search(mating_pool);    //use pair swap and 1-2 swap
        replacement(mating_pool, parent_pop);   //replace top 50 best to population
    }
    update_best_solution(parent_pop);   //update best solution
}
output_solution(best_sln, out_file);    //output the best solution
free_operaton();                        //free the memory


init_population(Ins(i), parent_pop){
    for each individual in parent_pop{
        //initialize all item as unpacked
        //randomly pack item into knapsack until the it violate the capacity
    }
}
```

```
selection(mating_pool, parent_pop){
    for each individual in mating_pool{
        //initialize this individual
        //select several candidates from parent_pop
        //packed the best candidate into mating_pool
    }
}

crossover(mating_pool){
    //use uniform crossover
    //crossover between the head and tale individual
}

mutation(mating_pool){
    for each individual in mating_pool{
        //mutate each chromosome with MUTATION_RATE
    }
}

feasibility_repair(mating_pool){
    for each individual in mating_pool{
        while(individual is not feasible){
            //find the packed item with minimal price
            //remove the item
            //evalutate the individual whether it violates the constrain
        }
    }
}

varaible_neighbourhood_search(mating_pool){
    for each individual in mating_pool{
        initialize current_solution
        while(neighbourhood < K){       //K: index of neibourhood
            neibourhood_solution = best_descent_vns(neibourhood_index, current_solu
tion)
            if(neibourhood_solution > current_solution){       //jump back to first
 neibourhood
                current_solution = neibourhood_index;
                neibourhood_index = 1;
            }
            else{       //go to next neibourhood
                neibourhood_index++;
            }
```

```
        }
        individual = current_solution
    }
}

best_descent_vns(neibourhood_index, current_solution){
    if(neibourhood_index == 1){     //pair swap
        //divide the packed and unpacked items into two lists storing it index for 
reducing the time complexity
        //travas two list to select two items to swap
        //if the swap is better then record this move
        //if the better swap is greater than the best swap, apply this swap
    }
    else if(neibourhood_index == 2){    //1-2 swap
        //divide the packed and unpacked items into two lists storing it index for 
reducing the time complexity
        //apply 1-
2 swap for 10,000 times to reduce the running time in one iteration
        //if the swap is better then record this move
        //if the better swap is greater than the best swap, apply this swap
    }
}

replacement(mating_pool, parent_pop){
    //joint mating_pool and parent_pop togeter
    //quick sort the mixed pool
    //select the top 50 solution as the parent_pop for next generation
}
```

3. The result of the algorithm:

Overall result(x2go):

file No.	1	2	3	4	5	6	7	8
Avg gap%	0.0273	0.1095	0.2619	0.1320	0.2439	0.3926	0.1917	0.3352
Max gap%	0.2092	0.3906	0.5926	0.5181	0.8557	0.9704	1.3877	0.8723
Min gap%	0.0000	0.0027	0.0454	0.0000	0.0175	0.0585	-0.0385	0.0149

Two instances are above 1% which is mknapcb7-02 and mknapcb7-07

One instance produces a better solution which is mknapcb7-12

Detail result(x2go):

5.100-00	24381	24330	0.002092	5.250-00	59312	59218	0.001585	5.500-00	120130	119619	0.005086	10.100-00	23064	23055	0.000390
5.100-01	24274	24274	0.000000	5.250-01	61472	61442	0.000488	5.500-01	117837	117212	0.005304	10.100-01	22801	22704	0.004254
5.100-02	23551	23538	0.000552	5.250-02	62130	61965	0.002656	5.500-02	121109	120602	0.005012	10.100-02	22131	22065	0.002982
5.100-03	23534	23493	0.001742	5.250-03	59446	59404	0.000707	5.500-03	120798	120148	0.005381	10.100-03	22772	22710	0.002723
5.100-04	23991	23966	0.001042	5.250-04	58951	58759	0.003257	5.500-04	122319	121636	0.005584	10.100-04	22751	22751	0.000000
5.100-05	24613	24613	0.000000	5.250-05	60056	59909	0.002448	5.500-05	122007	121284	0.005926	10.100-05	22777	22659	0.005181
5.100-06	25591	25591	0.000000	5.250-06	60414	60178	0.003906	5.500-06	119113	118485	0.005272	10.100-06	21875	21841	0.001554
5.100-07	23410	23410	0.000000	5.250-07	61472	61428	0.000716	5.500-07	120568	120024	0.004512	10.100-07	22635	22551	0.003711
5.100-08	24216	24216	0.000000	5.250-08	61885	61847	0.000614	5.500-08	121575	121018	0.004582	10.100-08	22511	22511	0.000000
5.100-09	24411	24360	0.002089	5.250-09	58959	58808	0.002561	5.500-09	120699	120200	0.004134	10.100-09	22702	22611	0.004008
5.100-10	42757	42757	0.000000	5.250-10	109109	108959	0.001375	5.500-10	218422	218035	0.001772	10.100-10	41395	41354	0.000990
5.100-11	42545	42545	0.000000	5.250-11	109841	109816	0.000228	5.500-11	221191	220668	0.002364	10.100-11	42344	42344	0.000000
5.100-12	41968	41959	0.000214	5.250-12	108489	108326	0.001502	5.500-12	217534	217093	0.002027	10.100-12	42401	42347	0.001274
5.100-13	45090	45090	0.000000	5.250-13	109383	109297	0.000786	5.500-13	223558	223050	0.002272	10.100-13	45624	45479	0.003178
5.100-14	42218	42218	0.000000	5.250-14	110720	110565	0.001400	5.500-14	218962	218498	0.002119	10.100-14	41884	41828	0.001337
5.100-15	42927	42927	0.000000	5.250-15	110256	110160	0.000871	5.500-15	220514	220031	0.002190	10.100-15	42995	42942	0.001233
5.100-16	42009	42009	0.000000	5.250-16	109016	108924	0.000844	5.500-16	219987	219639	0.001582	10.100-16	43559	43498	0.001400
5.100-17	45020	45000	0.000444	5.250-17	109037	108867	0.001559	5.500-17	218194	217715	0.002195	10.100-17	42970	42934	0.000838
5.100-18	43441	43441	0.000000	5.250-18	109957	109812	0.001319	5.500-18	216976	216323	0.003010	10.100-18	42212	42212	0.000000
5.100-19	44554	44554	0.000000	5.250-19	107038	106955	0.000775	5.500-19	219693	219442	0.001143	10.100-19	41207	41134	0.001772
5.100-20	59822	59822	0.000000	5.250-20	149659	149584	0.000501	5.500-20	295828	295621	0.000700	10.100-20	57375	57375	0.000000
5.100-21	62081	62081	0.000000	5.250-21	155940	155927	0.000083	5.500-21	308077	307937	0.000454	10.100-21	58978	58978	0.000000
5.100-22	59802	59802	0.000000	5.250-22	149316	149281	0.000234	5.500-22	299796	299578	0.000727	10.100-22	58391	58370	0.000360
5.100-23	60479	60479	0.000000	5.250-23	152130	152080	0.000329	5.500-23	306476	306228	0.000809	10.100-23	61966	61912	0.000871
5.100-24	61091	61091	0.000000	5.250-24	150353	150301	0.000346	5.500-24	300342	300110	0.000772	10.100-24	60803	60803	0.000000
5.100-25	58959	58959	0.000000	5.250-25	150045	149943	0.000680	5.500-25	302560	302344	0.000714	10.100-25	61437	61368	0.001123
5.100-26	61538	61538	0.000000	5.250-26	148607	148533	0.000498	5.500-26	301322	301105	0.000720	10.100-26	56377	56353	0.000426
5.100-27	61520	61520	0.000000	5.250-27	149772	149768	0.000027	5.500-27	306430	306204	0.000738	10.100-27	59391	59391	0.000000
5.100-28	59453	59453	0.000000	5.250-28	155075	155016	0.000380	5.500-28	302814	302569	0.000809	10.100-28	60205	60205	0.000000
5.100-29	59965	59965	0.000000	5.250-29	154662	154637	0.000162	5.500-29	299904	299705	0.000664	10.100-29	60633	60633	0.000000
avg			0.000273				0.001095				0.002619				0.001320
max			0.002092				0.003906				0.005926				0.005181
min			0.000000				0.000027				0.000454				0.000000

10.250-00	59187	58945	0.004089	10.500-00	117726	117130	0.005063	30.100-00	21946	21735	0.009615	30.250-00	56693	56307	0.006809
10.250-01	58662	58420	0.004125	10.500-01	119139	118321	0.006866	30.100-01	21716	21716	0.000000	30.250-01	58318	57929	0.006670
10.250-02	58094	57942	0.002616	10.500-02	119159	118343	0.006848	30.100-02	20754	20466	0.013877	30.250-02	56553	56272	0.004969
10.250-03	61000	60478	0.008557	10.500-03	118802	117773	0.008661	30.100-03	21464	21411	0.002469	30.250-03	56863	56367	0.008723
10.250-04	58092	57844	0.004269	10.500-04	116434	115399	0.008889	30.100-04	21814	21814	0.000000	30.250-04	56629	56189	0.007770
10.250-05	58803	58376	0.007262	10.500-05	119454	118658	0.006664	30.100-05	22176	22176	0.000000	30.250-05	57119	56753	0.006408
10.250-06	58607	58190	0.007115	10.500-06	119749	118587	0.009704	30.100-06	21799	21772	0.001239	30.250-06	56292	55857	0.007728
10.250-07	58917	58638	0.004735	10.500-07	118288	117419	0.007346	30.100-07	21397	21169	0.010656	30.250-07	56403	56138	0.004698
10.250-08	59384	59190	0.003267	10.500-08	117779	116888	0.007565	30.100-08	22493	22395	0.004357	30.250-08	57442	57167	0.004787
10.250-09	59193	58856	0.005693	10.500-09	119125	118170	0.008017	30.100-09	20983	20944	0.001859	30.250-09	56447	56004	0.007848
10.250-10	110863	110708	0.001398	10.500-10	217318	216707	0.002812	30.100-10	40767	40512	0.006255	30.250-10	107689	107341	0.003232
10.250-11	108659	108586	0.000672	10.500-11	219022	218227	0.003630	30.100-11	41304	41304	0.000000	30.250-11	108338	108068	0.002492
10.250-12	108932	108819	0.001037	10.500-12	217772	217209	0.002585	30.100-12	41560	41576	-0.000385	30.250-12	106385	106227	0.001485
10.250-13	110037	109805	0.002108	10.500-13	216802	216309	0.002274	30.100-13	41041	40972	0.001681	30.250-13	106796	106541	0.002388
10.250-14	108423	108325	0.000904	10.500-14	213809	213126	0.003194	30.100-14	40872	40845	0.000661	30.250-14	107396	107262	0.001248
10.250-15	110841	110571	0.002436	10.500-15	215013	214329	0.003181	30.100-15	41058	41058	0.000000	30.250-15	107246	107006	0.002238
10.250-16	106075	105946	0.001216	10.500-16	217896	217174	0.003314	30.100-16	41062	41038	0.000584	30.250-16	106308	105946	0.003405
10.250-17	106686	106506	0.001687	10.500-17	219949	219337	0.002782	30.100-17	42719	42692	0.000632	30.250-17	103993	103784	0.002010
10.250-18	109825	109554	0.002468	10.500-18	214332	213559	0.003607	30.100-18	42230	42230	0.000000	30.250-18	106835	106438	0.003716
10.250-19	106723	106522	0.001883	10.500-19	220833	220197	0.002880	30.100-19	41700	41700	0.000000	30.250-19	105751	105446	0.002884
10.250-20	151790	151714	0.000501	10.500-20	304344	303931	0.001357	30.100-20	57494	57494	0.000000	30.250-20	150083	150044	0.000260
10.250-21	148772	148746	0.000175	10.500-21	302332	302031	0.000996	30.100-21	60027	59997	0.000500	30.250-21	149907	149750	0.001047
10.250-22	151900	151747	0.001007	10.500-22	302354	302077	0.000916	30.100-22	58025	58013	0.000207	30.250-22	152993	152909	0.000549
10.250-23	151275	151170	0.000694	10.500-23	300743	300289	0.001510	30.100-23	60776	60776	0.000000	30.250-23	153169	153030	0.000907
10.250-24	151948	151860	0.000579	10.500-24	304344	304166	0.000585	30.100-24	58884	58884	0.000000	30.250-24	150287	150019	0.001783
10.250-25	152109	152005	0.000684	10.500-25	301730	301190	0.001790	30.100-25	60011	59902	0.001816	30.250-25	148544	148350	0.001306
10.250-26	153131	153059	0.000470	10.500-26	304949	304514	0.001426	30.100-26	58132	58132	0.000000	30.250-26	147471	147449	0.000149
10.250-27	153520	153461	0.000384	10.500-27	296437	296114	0.001090	30.100-27	59064	59064	0.000000	30.250-27	152841	152734	0.000700
10.250-28	149155	149113	0.000282	10.500-28	301313	301023	0.000962	30.100-28	58975	58887	0.001492	30.250-28	149568	149362	0.001377
10.250-29	149704	149576	0.000855	10.500-29	307014	306627	0.001261	30.100-29	60603	60603	0.000000	30.250-29	149572	149425	0.000983
			0.002439				0.003926				0.001917				0.003352
			0.008557				0.009704				0.013877				0.008723
			0.000175				0.000585				-0.000385				0.000149

4. tuning process:

First version:

MA was implemented with tournament selection, one-point crossover, feasibility repair that drop the cheapest items and first descent local search (only have pair swap). In this version, the result is quite bad that the average gap for mknapcb1 is about 5%. Although all the main parameters (CROSSOVER_RATE, MUTATION_RATE, POP_SIZE, TOURM_SIZE) were tuned, the result is still bad. The main reason is that the pair swap will end in local optimal with fixed number of items which cannot diversify the search for other optimal with different number of items. As a result, VNS is considered due to its variable neighborhood

Second version:

MA was implemented with tournament selection, uniform crossover, feasibility repair that drop the cheapest items then add back items and best_descent VNS (with 3 neighborhoods 1-swap, 2- swap, 3-swap). In this version, some idea from the paper in the coursework specification ("A genetic algorithm for the multidimensional knapsack problem") is implemented. Firstly, I use uniform crossover and add operation because it is implemented in the paper. Secondly, an additional add operation is implemented because it can improve the objective value after the drop operation in selection. Thirdly, VNS is implementing without shaking operation. In this case, this running time is too long for one iteration due to the best_descent VNS have too many for loops and the VNS will keep running until no more improvement is found. Initially, within one population, some of the solutions were randomly selected (about 10%) to perform VNS. After that, the iteration number for one instance increased (from less than 50 iteration to about 100 iteration) whereas the performance is not stable that some of the instance cannot go inside 1%. The reason for this is that for best_descent VNS cause a lot of time.

Final version:

MA was implemented with tournament selection, uniform crossover, feasibility repair that only drop the cheapest items and best_descent VNS (with 2 neighborhoods: 2- swap and 1-2-swap). In this version, add operation 1-swap and 2-1swap are removed because they cause too much time and will decrease the diversity of the population. Also, within one solution, a fixed number of 1-2swap is implemented that it will not go through all the feasible 1-2swap for one solution. In other words, this save a lot of time. More iteration can bring better performance.

Finally, the main parameter tuning will be discussed.

CROSSOVER_RATE: this parameter is often set to 0.5 in uniform crossover.

MUTATION_RATE: this is the way to diversity the population which is convergent. In this algorithm, it is set to 0.01 because a greater MUTATION_RATE will destroy the result got from VNS. When MUTATION_RATE is set to 0.05/0.1, the best solution is worse.

POP_SIZE: this is the main parameter effecting the running time for each iteration of one instance. In this algorithm, it is set to 50 because this brings the best performance. When POP_SIZE is set to 100/200, the time for on generation is too long so that the convergency speed is slow

TOURM_SIZE: this is the parameter effecting the convergent speed. If the TOURM_SIZE is too large, selection operation has higher rate to select items with large objective. As a result, to diversify the population, TOURM_SIZE is set to 2. When the TOURM_SIZE is set to 5/10, the convergent speed is too fast that it will be stuck into local optimal.

VNS_SWAP_NUM: this is the parameter decide how many swap will be considered in best_descent_vns(). In this algorithm, it is set to 10000 to exploit more swaps. When VNS_SWAP_NUM is set to 5000, the best solution is worse.

5. Pros and Cons of GA/MA methods

5.1. Pros for GA

1. The optimization process is applied on a set of individuals which means it have less chance to stuck into local optimal.
2. Have good performance on global search because of the crossover and selection operation. These two operations will retain good individuals

5.2. Cons for GA

1. Have bad performance on local search that the search efficiency is low in late generation.
2. Have many parameters to tune which is based on experience.

5.3.Pros for MA

1. Include both global and local search which will have better performance.

5.4.Cons for MA

1. Have many parameters to tune which is based on experience.