SOLVER IMPROVEMENT APPROACHES: MILES TO GO BEFORE WE SLEEP

BRANDON REESE
STEVE HARENBERG
LASZLO LADANYI
ROB PRATT
YAN XU

EURO MEETS NEURIPS 2022 VEHICLE ROUTING COMPETITION

IMPROVEMENT HIGHLIGHTS

Measurement and Benchmarking

Code Refactor and Optimization

Heuristic Improvements

MEASUREMENT AND BENCHMARKING

QUANTIFYING SOLUTION QUALITY AND COMPUTATIONAL PERFORMANCE

MEASUREMENT AND BENCHMARKING: IDEAS

- C++ profiling and static analysis

Function / Call Stack	CPU Time ▼	Instructions Retired	Microarchitecture Usage	Module	Function (Full)
▼ LocalSearch::MergeTWDataRecursive	12.515s	93,408,000,000	81.2%	genvrp	LocalSearch::MergeTWDataRecursive(TimeWindowData const&, TimeWindowData
[No call stack information]	12.515s	93,408,000,000	81.2%		
▼ LocalSearch::MoveSingleClient	3.580s	21,852,000,000	73.7%	genvrp	LocalSearch::MoveSingleClient(void)
[No call stack information]	3.580s	21,852,000,000	73.7%		
▶ LocalSearch::run	3.255s	13,740,000,000	48.2%	genvrp	LocalSearch::run(Individual*, double, double)
▶ std::_Rb_tree_increment	3.140s	3,432,000,000	11.0%	libstdc++.so.6.0.29	std::_Rb_tree_increment(std::_Rb_tree_node_base const*)
▶ LocalSearch::MoveTwoClients	2.990s	17,400,000,000	61.6%	genvrp	LocalSearch::MoveTwoClients(void)
▶ LocalSearch::SwapTwoClientsForOne	2.975s	15,468,000,000	60.5%	genvrp	LocalSearch::SwapTwoClientsForOne(void)
▶ LocalSearch::MoveTwoClientsReversed	2.785s	18,840,000,000	71.1%	genvrp	LocalSearch::MoveTwoClientsReversed(void)
▶ LocalSearch::SwapTwoSingleClients	1.890s	8,796,000,000	59.2%	genvrp	LocalSearch::SwapTwoSingleClients(void)
▶ LocalSearch::setLocalVariablesRouteV	1.835s	11,220,000,000	54.9%	genvrp	LocalSearch::setLocalVariablesRouteV(void)
▶ LocalSearch::TwoOptBetweenTrips	1.525s	8,676,000,000	62.5%	genvrp	LocalSearch::TwoOptBetweenTrips(void)
▶ LocalSearch::setLocalVariablesRouteU	1.480s	5,820,000,000	39.9%	genvrp	LocalSearch::setLocalVariablesRouteU(void)
▶ Matrix::get	1.220s	9,228,000,000	74.5%	genvrp	Matrix::get(int, int) const
▶ Individual::brokenPairsDistance	1.180s	2,460,000,000	20.7%	genvrp	Individual::brokenPairsDistance(Individual*)
▶ LocalSearch::SwapTwoClientPairs	1.105s	7,068,000,000	64.9%	genvrp	LocalSearch::SwapTwoClientPairs(void)
▶ LocalSearch::swapStar	0.720s	2,640,000,000	36.3%	genvrp	LocalSearch::swapStar(bool)
▶ std::__detail::_Synth3way::operator()<double, double>	0.640s	348,000,000	6.0%	genvrp	std::__detail::_Synth3way::operator()<double, double>(signed char, double const&, d
▶ _int_malloc	0.540s	1,596,000,000	34.5%	libc-2.17.so	_int_malloc

MEASUREMENT AND BENCHMARKING: IDEAS

- Achieving determinism

```
void Genetic::insertUnplannedTasks(Individual* offspring, std::unordered_set<int> unplannedTasks)
{
	// Initialize some variables
	int newDistanceToInsert = INT_MAX;	// TODO:
	int newDistanceFromInsert = INT_MAX;	// TODO:
	int distanceDelta = INT_MAX;		// TODO:

	// Loop over all unplannedTasks
	for (int c : unplannedTasks)
	{
		// Get the earliest and laster possible arrival at the client
		int earliestArrival = params->cli[c].earliestArrival;
		int latestArrival = params->cli[c].latestArrival;

		int bestDistance = INT_MAX;
		std::pair<int, int> bestLocation;

		// Loop over all routes
		for (int r = 0; r < params->nbVehicles; r++)
		{
			// Go to the next route if this route is empty
			if (offspring->chromR[r].empty())
			{
				continue;
			}
```

Undefined behavior:
Loop over unordered set
Integer overflow

```
newDistanceFromInsert = params->timeCost.get(c, offspring->chromR[r][0]);
if (earliestArrival + newDistanceFromInsert < params->cli[offspring->chromR[r][0]].latestArrival)
{
	distanceDelta = params->timeCost.get(0, c) + newDistanceToInsert
		- params->timeCost.get(0, offspring->chromR[r][0]);
	if (distanceDelta < bestDistance)
	{
		bestDistance = distanceDelta;
		bestLocation = { r, 0 };
	}
}

for (int i = 1; i < static_cast<int>(offspring->chromR[r].size()); i++)
{
	newDistanceToInsert = params->timeCost.get(offspring->chromR[r][i - 1], c);
	newDistanceFromInsert = params->timeCost.get(c, offspring->chromR[r][i]);
	if (params->cli[offspring->chromR[r][i - 1]].earliestArrival + newDistanceToInsert < latestArrival
		&& earliestArrival + newDistanceFromInsert < params->cli[offspring->chromR[r][i]].latestArrival)
	{
		distanceDelta = newDistanceToInsert + newDistanceFromInsert
			- params->timeCost.get(offspring->chromR[r][i - 1], offspring->chromR[r][i]);
		if (distanceDelta < bestDistance)
		{
			bestDistance = distanceDelta;
			bestLocation = { r, i };
		}
	}
}
```

MEASUREMENT AND BENCHMARKING: IDEAS

- Achieving determinism

Undefined behavior resolved after refactor!

```
void Genetic::insertUnplannedTasks(Individual* offspring, std::unordered_set<int> unplannedTasks)
{
    // Initialize some variables
    int newDistanceToInsert = INT_MAX;     // TODO:
    int newDistanceFromInsert = INT_MAX;   // TODO:
    int distanceDelta = INT_MAX;           // TODO:

    // Loop over all unplannedTasks
    for (int c : unplannedTasks)
    {
```

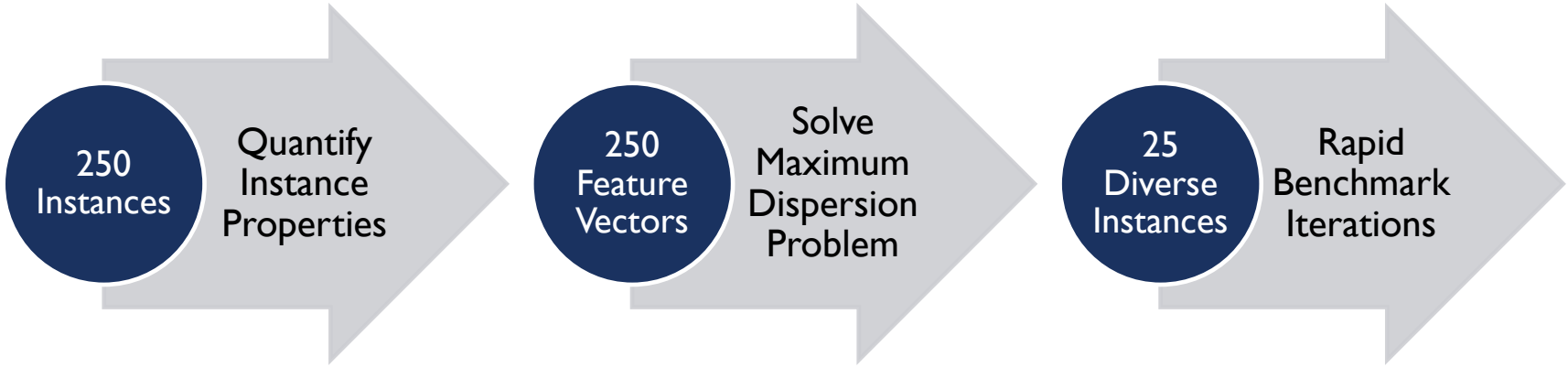
```
void Genetic::insertUnplannedTasks2(Individual* offspring, std::vector<bool> unplannedTasks)
{
    // Initialize some variables
    int newDistanceToInsert;      // Initialized for each route
    int newDistanceFromInsert;    // Initialized for each route
    int distanceDelta;            // Initialized for each route

    // Loop over all unplannedTasks
    int c=0;
    for (bool b : unplannedTasks)
    {
        if(!b){
            c++;
            continue;
        }
```

```
        for (int i = 1; i < static_cast<int>(offspring->chromR[r].size()); i++)
        {
            newDistanceToInsert = params->timeCost.get(offspring->chromR[r][i - 1], c);
            newDistanceFromInsert = params->timeCost.get(c, offspring->chromR[r][i]);
            if (params->cli[offspring->chromR[r][i - 1]].earliestArrival + newDistanceToInsert < latestArrival
                && earliestArrival + newDistanceFromInsert < params->cli[offspring->chromR[r][i]].latestArrival)
            {
                distanceDelta = newDistanceToInsert + newDistanceFromInsert
                    - params->timeCost.get(offspring->chromR[r][i - 1], offspring->chromR[r][i]);
                if (distanceDelta < bestDistance)
                {
                    bestDistance = distanceDelta;
                    bestLocation = { r, i };
                }
            }
        }
```

```
        for (int i = 0; i <= static_cast<int>(offspring->chromR[r].size()); i++)
        {
            int previousOnRoute = (i == 0) ? 0 : offspring->chromR[r][i - 1];
            int currentOnRoute = (i == static_cast<int>(offspring->chromR[r].size())) ? 0 : offspring->chromR[r][i];
            newDistanceToInsert = params->timeCost.get(previousOnRoute, c);
            newDistanceFromInsert = params->timeCost.get(c, currentOnRoute);
            if (params->cli[previousOnRoute].earliestArrival + newDistanceToInsert < latestArrival
                && earliestArrival + newDistanceFromInsert < params->cli[currentOnRoute].latestArrival)
            {
                distanceDelta = newDistanceToInsert + newDistanceFromInsert
                    - params->timeCost.get(previousOnRoute, currentOnRoute);
                if (distanceDelta < bestDistance)
                {
                    bestDistance = distanceDelta;
                    bestLocation = { r, i };
                }
            }
        }
```

MEASUREMENT AND BENCHMARKING: IDEAS

- Representative subsets of benchmark instances



Feature set:

- dimension
- capacity
- demandMeanOverCapacity
- distanceToDepotCV
- demandCV
- timeWindowCV
- serviceCV

CODE REFACTOR AND OPTIMIZATION

ROAD TO STATIC SCORE IMPROVEMENT: SPEED THINGS UP WHEREVER POSSIBLE

CODE REFACTOR AND OPTIMIZATION: IDEAS

- Avoid recursions and force inline execution of simple functions
- Rewrite Matrix class to speed up Matrix::get(), which is called all over the place
- Swap out data structures and sort algorithms for more efficient ones

CODE REFACTOR AND OPTIMIZATION: IDEAS

- Avoid recursions and force inline execution of simple functions
- Rewrite Matrix class to speed up Matrix::get(), which is called all over the place
- **Swap out data structures and sort algorithms for more efficient ones**

```
57  -         std::multiset<std::pair<double, Individual*>> indivsPerProximity;
      // The other individuals in the population (can not be the depot 0), ordered by
      increasing proximity (the set container follows a natural ordering based on the
      value of the first pair)
```

```
58  +         std::multiset<double> proximities;                    // The proximities
      of other individuals in the population (can not be the depot 0), ordered by
      increasing proximity (the set container follows a natural ordering based on the
      value of the first pair)
59  +         std::unordered_map<Individual*,double> proximityPerIndividual;
```

CODE REFACTOR AND OPTIMIZATION: IDEAS

- Avoid recursions and force inline execution of simple functions
- Rewrite Matrix class to speed up Matrix::get(), which is called all over the place
- **Swap out data structures and sort algorithms for more efficient ones**

16 baselines/hgs_vrptw/Individual.cpp

```
@@ -115,15 +115,7 @@ void Individual::shuffleChromT()
```

```
115
116  void Individual::removeProximity(Individual* indiv)
117  {
118 -        // Get the first individual in indivsPerProximity
119 -        auto it = indivsPerProximity.begin();
120 -        // Loop over all individuals in indivsPerProximity until indiv is found
121 -        while (it->second != indiv)
122 -        {
123 -                ++it;
124 -        }
125 -        // Remove indiv from indivsPerProximity
126 -        indivsPerProximity.erase(it);
127  }
```

```
115
116  void Individual::removeProximity(Individual* indiv)
117  {
118 +        proximities.erase(proximityPerIndividual[indiv]);
119  }
```

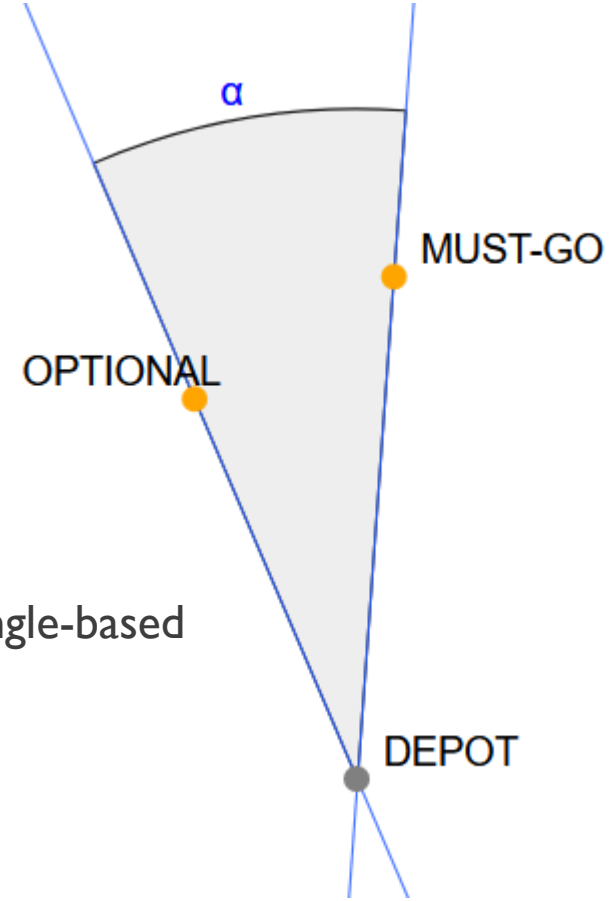
HEURISTIC IMPROVEMENTS

SIMPLE STRIDES TOWARD A BETTER DYNAMIC SOLVER

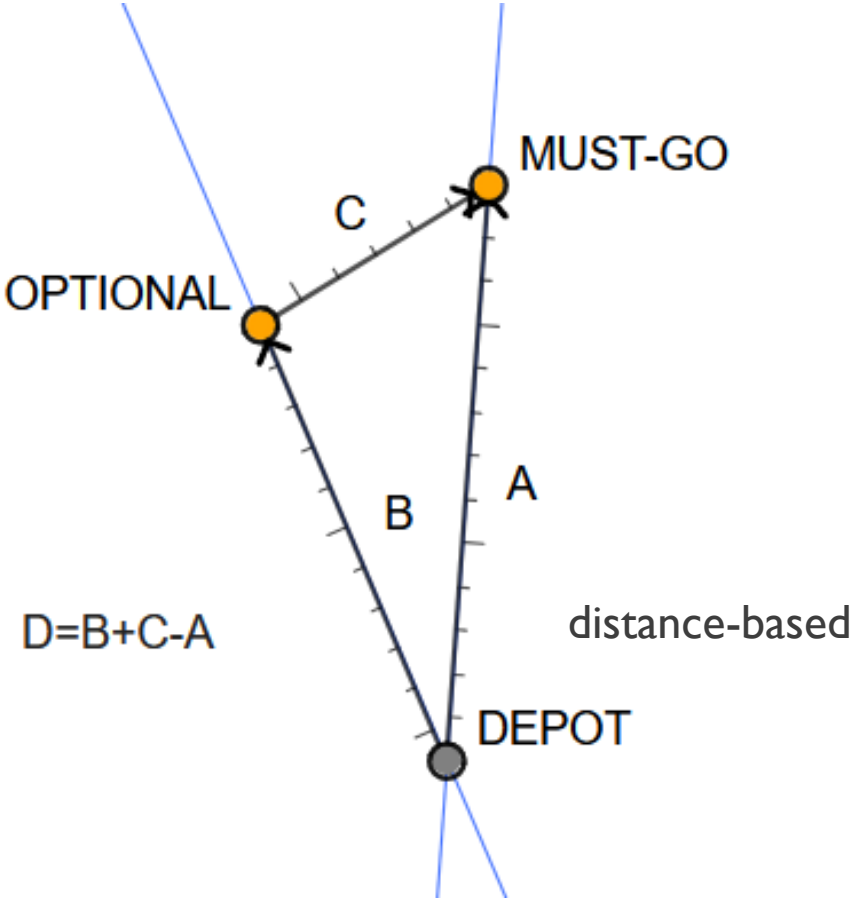
HEURISTIC IMPROVEMENTS: IDEAS

- Postprocess: if a route contains 100% optional clients, postpone them all
- Preprocess: mark only a subset of optional clients for potential dispatch
  - choose this subset by ranking all clients via one of two simple heuristics, then:
    - select all clients with heuristic score less than a fixed parameter OR
    - select a percentage of all optional clients with smallest heuristic scores, based on a fixed parameter OR
    - select up to the N optional clients with smallest heuristic scores, where N is a multiple of the number of must-go clients

HEURISTIC IMPROVEMENTS: IDEAS



angle-based



distance-based

The two heuristics

Github:
https://github.com/steveharenberg/sas-euro-neurips-vrp-2022.git

THANK YOU